PROACTIVE ENERGY MANAGEMENT FOR SMART BUILDING AND COMPUTE SERVER ARCHITECTURES

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Abhinandan Majumdar

December 2017

PROACTIVE ENERGY MANAGEMENT FOR SMART BUILDING AND COMPUTE SERVER ARCHITECTURES

Abhinandan Majumdar, Ph.D.

Cornell University 2017

The growing demand for improved operational performance, coupled with the increasing scarcity of energy resources, calls for new approaches to improving the energy efficiency of smart buildings and computer systems. Traditional energy management techniques have been either *reactive* or *locally predictive* at best. These schemes often underperform, by either failing to meet the desired performance target or consuming excess energy. Moreover, different applications and environments create a diverse set of challenges. Therefore, there is a dire need to develop new techniques that approach energy-performance optimality under stringent and diverse application and environmental conditions.

In this dissertation, we propose *proactive* management techniques for Heating, Ventilation, and Air-Conditioning (HVAC) in smart buildings, and for dynamic power management of heterogeneous processors. We show how the lack of future visibility and adaptivity of traditional energy management techniques proposed in these two domains degrades energy and performance. We develop proactive techniques to improve energy efficiency in buildings and processors.

We first focus on building energy management. We propose to automatically assign meetings to rooms to reduce overall energy consumption. We derive an HVAC energy model for meeting assignment by characterizing building energy behavior. Using this energy model, we propose several assignment algorithms, and analyze their optimality and scalability. We also characterize how different

factors impact energy savings, when it is worthwhile to use complex assignment algorithms, and when simpler methods suffice. We further extend this model to include weather factors, and develop a methodology for assigning meetings to the most appropriate room given the expected weather.

We then propose to apply Model Predictive Control (MPC) to dynamically balance HVAC energy consumption and occupant comfort. Traditional energy management techniques ignore past discomfort behavior and therefore poorly trade off energy for comfort. Our MPC framework uses a probabilistic model to predict the upcoming occupancy and discomfort history, in order to adaptively balance energy consumption and occupant comfort. Our approach achieves high energy efficiency while operating within a specified discomfort target.

For heterogeneous processors, we propose MPC-based dynamic General Purpose Graphics Processing Unit (GPGPU) power management. Traditional schemes ignore future kernel behavior, and may degrade performance and energy efficiency due to their inability to plan for the performance and energy characteristics of future phases. Our approach proactively configures hardware states based on recent execution history, the pattern of upcoming kernels, and the predicted behavior of those kernels, and adaptively varies its future visibility in order to achieve high energy savings with negligible performance impact and overhead.

We extend this framework for workloads that use the CPU and the GPU concurrently. Our MPC approach simultaneously optimizes the CPU and GPU across adaptively-managed time windows consisting of multiple phases of CPU and GPU applications. We explore several alternatives that trade off future visibility for computational overhead, and demonstrate significant energy savings over current state-of-the-art approaches.

BIOGRAPHICAL SKETCH

Abhinandan Majumdar hails from an Indian city called Bilaspur, Chhattisgarh and was raised in a Bengali family. He joined National Institute of Technology Karnataka (NITK) at Surathkal to pursue an undergraduate degree in Computer Science and Engineering. During his undergraduate studies, he developed interests in computer architecture and digital hardware design. He finished his undergraduation in 2006 and joined Intel, Bangalore as a software engineer in the Mobility Group. At Intel, he performed driver development for Intel Turbo Memory.

After a year at Intel, Abhinandan joined Columbia University to pursue a Master of Science in Computer Engineering. At Columbia, he focused on embedded systems, VLSI and analog systems. After his Masters, Abhinandan worked as a research assistant at NEC laboratories from 2009 to 2011, where he architected an FPGA-based massively parallel accelerator for machine learning applications.

In 2011, he joined Cornell University's Computer Systems Laboratory under the mentorship of Prof. David H. Albonesi. He worked on smart buildings and proposed techniques to improve the energy efficiency of buildings while making occupants comfortable. In 2015, he interned at the research division of Advanced Micro Devices (AMD), where he applied Model Predictive Control (MPC) to dynamically manage GPGPU power. Following his return to Cornell University, he resumed his research on proactive energy management for smart buildings and heterogeneous processors.

Abhinandan defended his Ph.D. dissertation in August 2017 and plans to join Intel Corporation in Hillsboro, OR as a power management architect in the Platforms Engineering Group.

This dissertation is dedicated to my parents, my sister and to my wife Rudraxi.

Mamma (Mom), hope I made you proud!

ACKNOWLEDGEMENTS

This dissertation would have been impossible without the continuous support, patience and encouragement of my advisor Prof. David H. Albonesi. His outstanding technical advice and inspiring words, such as to try my hardest, make innumerable attempts to achieve the best results, never lose focus on the big picture, never compromise integrity; are a few things I will always remember from my Ph.D. Furthermore, his backing during my internship with AMD allowed me to narrow the bridge between buildings and processors, which in turn helped me to make this dissertation stronger. His support for my teaching aspirations, his feedback during my teaching assistantship, and his advice on polishing my soft skills helped me win the outstanding teaching assistant award. His tireless edits of my writings not only taught me the nuances of articulating an idea succinctly and coherently, but also helped me to produce publications of good quality. Finally, his advice on balancing my academic obligations and personal life made me thoroughly enjoy my time doing Ph.D. at Cornell.

I would also like to thank my committee members for being an integral part of my dissertation. Apart from their feedback during my proposal and defense examination, every committee member contributed uniquely to my research. For instance, Prof. Brandon M. Hencey taught me control theory and model predictive control (MPC), which became the foundation of my *future-aware* proactive energy management work for buildings and processors (Chapters 5, 6, and 7). His advice of formulating the problem mathematically before coming up with solutions, inspired me to approach the problem differently than the conventional approach followed by computer architects. Prof. Zhiru Zhang encouraged me to think beyond the heuristic I developed for my very first paper on energy-aware meeting assignment. His advice on formulating the problem

in an Integer Linear Programming model, and developing a model based on building power characterization helped me to propose an energy savings model for meeting room assignment and taking it further to include the solar factors. These two ideas became the basis for Chapters 3 and 4. Prof. Bart Selman taught me Artificial Intelligence, and his class lecture on A* search algorithm inspired me to come up with *BT-reduced* algorithm, presented in Chapter 4, to cut down the exhaustive search cost of an oracle algorithm. Finally, this dissertation would have been impossible without the support of Dr. Indrani Paul. Her encouragement to incorporate my AMD internship work into my Ph.D. helped me to make this dissertation stronger and relevant to two different fields. Her optimistic view towards research, her advice on understanding every minute details of the work, and coming up with an alternative plan when things did not proceed as expected, helped me to overcome the obstacles I faced during my collaboration with AMD.

I would also like to thank my colleagues at AMD Research, namely, Dr. Joseph L. Greathouse, Dr. Wei Huang, and Dr. Leonardo Piga for their feedback to make my dissertation better, and for always inspiring me to think ideas that are practical in nature. It was their encouragement along with Dr. Paul's support that inspired me to apply the MPC idea on GPGPU power management and submit it to a conference that was due within a month. This work ultimately became the Chapter 6 of my dissertation.

Furthermore, I would like to thank Dr. Srihari Cadambi from Google, and Dr. Srimat Chakradhar from NEC Laboratories for believing in me and for encouraging me to pursue a Ph.D. degree. Both of them have been influential in honing my analytical skills and for providing me all the necessary platform to achieve my goals.

From the Computer Systems Laboratory (CSL), I would like to thank my officemates Skand Hurkat, Neeraj Kulkarni and Nitish Srivastava for their help on many technical challenges I faced during my Ph.D. and for keeping the office a fun place to work. I would also like to thank the other two musketeers of our trio – Shreesha Srinath and Xiaodong Wang for being great friends. I would also like to thank the CSL faculty and the students for giving me a sense of community and making a great place to do research.

Within Cornell, I would like to thank Mr. Scott Coldren for handling all the administrative issues. It was Scott who encouraged me to apply to Cornell, for which, I will remain indebted to him. I would also like to thank Prof. David Way and Cornell's Center for Teaching Excellence (CTE) for giving me a different perspective on teaching. It was their techniques that I practically applied during my teaching assistantship, which gave me department-wide recognition.

I would also like to thank Somenath, Vibha and Chintan for being great friends for the last ten years, for giving me a reason to step out of Ithaca once in a while, and for helping me during my internship days at Austin. Special thanks to Chintan and Bhargav for helping me with my statement of purpose, which got me admitted to the Ph.D. program of Cornell.

Finally, I would like to thank my parents, my in-laws and my sister for believing in me when I did not, for encouraging me to look at the bigger picture and not get overwhelmed by failures, and most importantly, for supporting my ambition of pursuing a doctorate degree. I would especially like to thank my mother for insisting me to get the highest educational degree.

Lastly, this acknowledgment section would be incomplete without thanking my wife Rudraxi. It was her love, her belief in my abilities, and her support and patience, which helped me to finish this dissertation. Love you, Rudra!

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

MOTIVATION AND RESEARCH OUTLINE

Worldwide energy is fast becoming a scarce resource. Eighty-one percent of world energy depends on fossil fuels, and these energy resources are expected to exhaust within the next 50-200 years, while the energy demand is estimated to double by 2050 [8]. For instance, building operations in 2010 consumed 11 trillion kWh of energy, constituting 41% of the total US energy. Seventy-four percent of this energy is derived from fossil fuels and building energy demand is projected to grow by 20% by 2035 [1]. Data centers, on the other hand, consumed 91 billion kWh of energy in 2013, and this energy usage is expected to grow by another 50% by 2020 [18]. Recently, the Department of Energy has declared achieving exascale performance under 20 MW of power as one of their top ten exascale research challenges [19], while President Obama authorized an executive order to achieve this goal by 2023 [7, 71]. The growing demand for improved system performance along with depleting energy resources has created an indispensable need for new solutions to energy management.

Traditional energy management techniques have been either *reactive* or *locally predictive* at best. Reactive schemes change the control parameters upon observing an input change, while locally predictive techniques use the immediate past history to adjust the control parameters. These schemes often underperform or consume excess energy. *Proactive* techniques, on the other hand, nimbly change the inputs to optimally use the available system resources, take appropriate anticipatory actions by estimating the future input changes, or adapt future decisions based on past performance behavior. These schemes have the potential to significantly reduce energy, while achieving the desired system per-

formance.

Previously proposed anticipatory techniques have been effective for applications capable of tolerating the overheads associated with complex optimizations and implementation-level challenges. For instance, anticipatory policies in building power management [46, 60, 132], urban traffic control [56, 88], process control for chemical plants [109], and biological processes [114] benefit from complex optimizations to achieve high optimality. These approaches, however, do not adapt based on past performance, and lack the ability to dynamically change future decisions. Moreover, applications such as processor-level energy management often involve determining the optimal hardware settings at millisecond timescale by a control policy running on an embedded microcontroller. As a result, they rely on locally predictive techniques. To date, the use of proactive energy management techniques has received little attention.

This dissertation proposes proactive techniques to reduce system energy consumption while meeting the desired system performance. We focus on Heating, Ventilation, and Air-Conditioning (HVAC) in buildings, and power management in heterogeneous processors. We show how proactive techniques driven by modeling the system and input behavior along with appropriately adjusting the algorithms can lead to higher energy savings in a practical way.

1.1 Buildings and Processors

Both buildings and processors share many commonalities. For instance, for HVAC energy management, we consider the building as the *system*, the expected occupancy as *input*, weather factors as *disturbances*, the zone set tempera-

ture and ventilation air flow rate as the *control parameters*, and the occupant comfort as desired *system performance*. For processor-level power management, we consider the CPU and the GPU as the *system*, applications as *input*, the system-level dynamic voltage frequency scaling (DVFS) states and the number of active processing units as *control* parameters, impact in power and performance due to thermals and shared resources as *disturbances*, and finally, the application throughput as desired *system performance*.

Modeling the system and the upcoming input behavior often require developing prediction models using techniques such as machine learning, system identification, physical modeling, or intuitions derived from offline experimental characterization. These models can be either trained *offline* or can be fine-tuned *online* during the system operation. However, obtaining close to global optimality while working with inaccurate prediction models is not always achievable. Adding feedback may improve optimality but may lead to non-convergence or oscillations. Algorithmic challenges further make implementing complex optimization algorithms difficult. Given these many design choices, it is generally not well understood what methods are suitable to achieve optimality for different application challenges.

Figure 1.1 presents a mapping between application characteristics and suitable design choices. Applications that are inherently stable and behave deterministically may not require feedback, and can benefit from offline model training. On the contrary, if the model is highly nonlinear and uncertain, feedback helps fine-tune the system performance towards optimal operation, while online models adapt to new observations captured during runtime. Algorithmic or implementation challenges also determine what optimization algorithms to

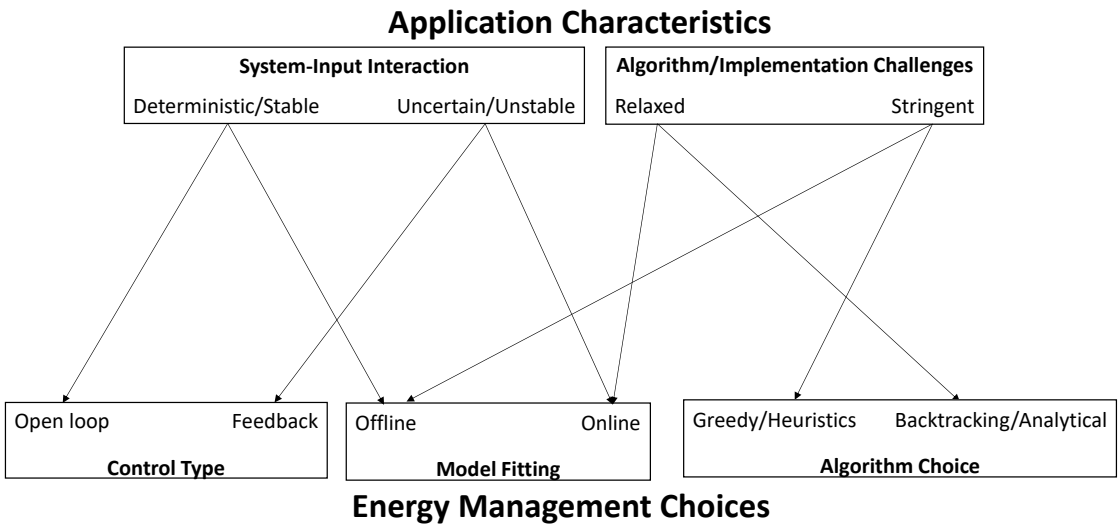


Figure 1.1: Suitable design choices for different application characteristics.

select. For less demanding algorithmic or implementation requirements, the application can benefit from complex optimizations such as backtracking or analytical solutions. However, applications working under stringent time requirements may rely on greedy heuristics.

Despite the similarities between buildings and processors, their intrinsic implementation-level challenges demand suitable adjustments in their respective techniques to avoid impacting the operational efficiency. Buildings take a relatively long time to respond to a particular control action, and the Building Automation System has a dedicated high-performance computer to determine energy optimal control decisions. As a result, buildings can benefit from online learning and complex optimization algorithms. However, uncertainty in inputs and disturbances, and complex building dynamics, make modeling the interaction between the system and input challenging, and can therefore benefit from feedback. Processors, on the other hand, must respond within milliseconds to a particular control setting. As a result, the use of complex optimization solvers along with online training models is an ambitious proposition. Furthermore, the

complex interaction of processors with a wide range of inputs make it challenging to accurately model its energy and performance behavior. For these reasons, prior work primarily leverages offline trained models or intuitions from characterization, and use rule-based policies or greedy optimizations to determine the appropriate hardware settings. Rule-based reactive policies, however, are ineffective for building HVAC automation because the slow response of buildings leads to occupant discomfort.

In this dissertation, we show how proactive schemes improve the energy efficiency of both buildings and processors by nimbly changing the inputs, adapting from the continuous feedback and looking ahead in the future to influence the current decision. Our proactive mechanisms use models that capture the underlying *system* behavior and its interaction with the *inputs* and *disturbances*, estimation models or profiled data to predict the future application behavior, and *open-loop* or *feedback* control to dynamically change the decisions based on runtime performance. We make suitable adjustments to our proactive methods to make them practical for both the specific domain (buildings or processors) while considering their respective implementation-level challenges.

1.2 Outline of the Dissertation

In Chapter 2, we present related work in building and processor power management. Then we address proactive techniques to reduce the HVAC energy consumption in buildings. In Chapter 3, we explore automatic assignment of meetings to specific rooms to reduce the HVAC energy consumption. We propose several room assignment algorithms such that size mismatches and timing con-

flicts are avoided. Since these conflicts can be extracted deterministically from the meeting schedule itself, we propose open-loop control policies. We characterize the building power behavior during meeting times and gaps through EnergyPlus, and derive an energy model. Using this model, we develop several search algorithms, and an Integer Linear Programming (ILP) formulation. We also characterize how various meeting, room, and weather related factors affect the energy savings, and when is it worthwhile to use complex assignment algorithms. We prove that finding a feasible assignment itself is NP-complete. As a result, backtracking search algorithms do not scale for larger problems, while greedy methods have high scalability with some loss in optimality. The ILP formulation leverages fast solvers, and thus shows better scalability than backtracking methods.

In Chapter 4, we extend our meeting assignment model to include solar factors. While our proposed energy model from Chapter 3 is effective for various meetings and room characteristics, such as meeting gaps, occupancy and room size, we observe that rooms may differ in their energy characteristics depending on their orientation toward the sun. We perform initial experiments that show that scheduling a meeting to a south-facing room might save significant energy compared to scheduling it in a north-facing room, despite the former being relatively bigger in size. However, such behavior is observed only on certain days. Motivated by this finding, we include weather parameters, such as sky visibility and incident solar radiation, to determine when the bigger south-facing room consumes less meeting energy than the north-facing room and by what magnitude. We include a solar factor in our energy model, and develop a methodology of learning the relationship between weather parameters and the solar factor. We show that our solar-aware energy model can intelligently place

meetings to rooms based on expected weather conditions for a wide range of meetings and buildings.

In Chapter 5, we propose a model predictive control (MPC) technique to dynamically balance HVAC energy consumption and occupant comfort. The MPC algorithm optimizes energy over a future window of timesteps and dynamically adapts its behavior according to past discomfort performance. Previous work has jointly minimized energy and instantaneous discomfort, and has ignored past discomfort behavior in future energy optimization decisions. This results in reduced energy savings or occupant comfort. Our method uses discomfort history, and plays the dual role of saving energy when the discomfort is smaller than the target budget, and maintaining comfort when the discomfort margin is small. Our MPC framework uses a probabilistic model to predict the upcoming occupancy, a state space model to capture the building dynamics, and feedback to adaptively balance energy consumption and occupant discomfort. For a variety of occupancy profiles, our MPC method saves energy while operating within the allowed discomfort target.

In Chapter 6, we focus on dynamic GPGPU power management in heterogeneous processors. We use an approximation of MPC to improve the energy efficiency of GPU kernels without compromising performance. Traditional schemes that statically optimize individual kernels, or dynamically optimize over multiple iterations of each kernel, ignore future behavior, and therefore can degrade performance and energy due to their inability to plan for the performance and energy characteristics of upcoming kernels. Our system proactively configures the hardware states based on recent execution history, the pattern of upcoming kernels, and the predicted behavior of those kernels, and adaptively

tunes its future visibility to achieve high energy savings with negligible performance impact and overhead. Our framework uses performance and power prediction models, a kernel pattern extractor to predict the upcoming kernels, and feedback to dynamically recover from the inaccuracies associated with the prediction models. To limit the overhead, our MPC scheme accounts for the real-time MPC optimization overheads along with the performance impact due to MPC decisions to vary the future visibility. The exponential search space of the problem, and stringent timing requirements make full-scale implementation of MPC impractical. As a result, we create greedy and heuristic approximations of MPC, and achieve significant energy savings with negligible performance impact over both state-of-the-practice and state-of-the-art approaches.

In Chapter 7, we extend our MPC-based power management infrastructure for heterogeneous applications that use the CPU and the GPU simultaneously. We create heterogeneous workloads using SPEC and GPGPU benchmarks. Our MPC-approach simultaneously optimizes the CPU and the GPU across adaptively managed time windows to improve the application-level energy efficiency with low performance impact. Our power manager consists of a predictor that estimates the performance and power of future time windows, an optimizer and profiled baseline performance data. Our scheme adaptively looks into the future time windows of the application, and builds performance credit to determine the appropriate setting of the upcoming application window. We explore several alternatives that trade off future visibility with computational overhead, and demonstrate significant energy savings over current state-of-the-art approaches.

Finally, we discuss possible extensions of this work in Chapter 8 and con-

clude in Chapter 9.

CHAPTER 2

RELATED WORK

2.1 Energy-efficient Meeting Room Assignment

The work by Pan et al. [126] is the first research to our knowledge that proposes intelligent meeting room assignment algorithms to reduce the building energy use. The authors use an energy-temperature correlation model and propose a greedy method to statically assign meetings to closely-sized rooms. This algorithm closely resembles our *Capacity Match* algorithm presented in Chapter 3. Several other works have proposed an Integer Linear Programming (ILP) formulation to solve the meeting room assignment problem [32, 84, 97, 98, 125]. In Chapter 3, we derive an energy model from a building power characterization study and propose several search algorithms and an ILP formulation.

In terms of modeling building energy for meeting room assignment, Pan et al. use EnergyPlus to model the building thermal behavior of a building [125] in order to reduce the building electric cost in a dynamic power market. Others have used a Resistor-Capacitor network to model the building thermal behavior [66, 97, 98]. These works, however, do not identify the important energy factors affecting the energy savings potential.

Lastly, Kwak et al. propose an energy model that is independent of occupancy factors and considers no energy when a room is unoccupied [84]. Chai et al. consider the extra energy spent before a meeting is scheduled, but their energy model is occupancy independent [32]. We show that these assumptions impact the model accuracy, while our more detailed model closely matches the

savings obtained from EnergyPlus.

2.2 Solar Aware Energy-Efficient Meeting Assignment

Several authors have studied the impact of solar factors on building HVAC energy. Laur studies the impact of solar radiation on a university hall [87]. Royer et al. use system identification to capture the building dynamics and its impact from the solar radiation using EnergyPlus simulations [140], while Fumo et al. extract coefficients from EnergyPlus simulations to estimate the energy consumption [61]. Maasoumy et al. estimate the building states and parameters using solar factors to perform predictive HVAC control [106, 105], while Yan et al. model occupant behavior and consider the solar gains in their building energy simulations [171]. Corbin and Henze apply Model Predictive Control (MPC) to predictively control residential HVAC by considering the solar gains of the walls of their building surface [38], while Oldewurtel et al. use MPC along with weather forecasts and incoming solar radiation to efficiently control HVAC operations [121]. Kleiminger et al. present a simulation framework, which considers the heat gains due to solar radiation, to evaluate the energy savings potential of occupancy prediction algorithms [78], while Sharmin et al. propose a framework to identify occupant activities impacting building energy [144], and use it with an occupant prediction model to reduce the heating energy [145]. These two works conclude that the south-facing rooms of a building situated in the northern hemisphere require considerably less heating energy than north-facing rooms.

Among the studies on how solar radiation impacts meeting assignments and

its associated energy savings, Lim et al. consider solar gains in their Resistor-Capacitor based energy model [96, 97, 98]. However, their model considers a range of solar gain and does not identify how weather conditions impact the solar gain. Similarly, Pan et al. rely on EnergyPlus simulations to assign meetings, which implicitly captures the solar gains from weather data [125]. This work does not identify which solar-related energy factors impact the meeting room energy savings, and how they impact scheduling decisions. In Chapter 4, we enhance our proposed energy model to include a solar factor, which uses the heterogeneity among the rooms based on how they are oriented and their respective capacities. Our assignment algorithms using our proposed energy model identify the times within a day when it is efficient to assign meetings to a bigger south-facing room versus a smaller north-facing room, and vice versa.

2.3 Dynamic HVAC Energy and Occupant Comfort Optimization

Optimizing HVAC energy using real-time occupancy detection from a plurality of sensors has been proposed by Agarwal et al. [21]. Reactive policies based only on occupancy detection are rule-based or reactive at best, and either waste energy or sacrifice comfort. Motivated by this limitation, occupancy prediction models, such as autonomous agent models [55, 95], sensor-utility network [115], naïve Bayes classifier [160], probabilistic graphic models [47], Markov chains [46, 54, 122], hidden Markov model [73, 85, 100], support vector machines [108], decision trees [173] and neural networks [51, 172] have been proposed. Among them, [46, 54, 100, 108, 172, 173] have used occupancy prediction to optimize

HVAC energy while satisfying occupant comfort requirements. However, none of these works consider the ramification of occupancy mispredictions on energy and discomfort during actual system operation.

Furthermore, recent works have used instantaneous discomfort values to balance energy and discomfort. For instance, [46, 60, 132] use MPC algorithms to jointly optimize energy and comfort by using a scalar parameter to prioritize energy versus comfort, while [53, 83, 117] use heuristic algorithms. Both approaches require operators to manually tune the parameters at the time of synthesis. These parameters, once fixed, cannot adapt to the dynamic behavior of the system during operation. Our method does not rely on any parameter tuning. Rather, it dynamically adjusts its future decisions based on the past discomfort performance. Minimizing energy by constraining the instantaneous discomfort to a certain limit is considered in [22, 48, 64, 101, 119, 121]. Since discomfort is felt over time, constraining instantaneous discomfort is not a true representation of occupant comfort. This also presents fewer opportunities to make future energy-efficient decisions based on the past energy-discomfort performance.

2.4 Dynamic GPGPU Power Management

For processor-level power management, there has been research [24, 91, 102, 162] that distributes workloads among the CPU and GPU to increase the power efficiency of GPGPU applications. Other work [26, 67, 111, 128, 130, 143] reconfigures hardware through dynamic voltage and frequency scaling, while others [23, 163, 165] use power gating. The authors of [37, 86, 110, 177] optimize per-

formance under a power cap, while the authors of [63, 129] optimize under a thermal constraint. None of these works consider kernel-level heterogeneity and future kernel behavior.

Among the power-performance optimization studies based on predictive models, there has been research [70, 75, 150] that proposes analytical estimation models, while other work [44, 89, 170, 176] presents learning or statistical models. The use of estimation models to dynamically optimize power efficiency has been proposed by [40, 41, 44, 151]. The work of [143] monitors performance counters and dynamically tunes the GPU knobs to operate under *performance boost* or *energy efficiency* mode, while other work [130] predicts the application sensitivities of the hardware configuration to determine an energy-optimal configuration. All of these works are solely *predictive* schemes and lack adaptivity to fine-tune future decisions based on past performance changes.

Within the adaptive power management schemes, Paul et al. [128] use linear regression models to predict performance and power sensitivities and use two levels of tuning to adapt according to the past performance trend at the kernel level, without across-kernel considerations. In Chapter 6, our baseline *PPK* scheme represents such state-of-the-art future agnostic schemes.

2.5 Dynamic Power Management of Heterogeneous Architectures

Dynamic power management of heterogeneous architectures involves co-scheduling application on the most appropriate nodes and varying the power

states to improve energy efficiency. Heterogeneity may be at the ISA level [80, 127, 130], among the nodes of a cluster [118, 134, 175], or may arise due to process variation or defects within the chip [155, 167].

Resource-aware scheduling of applications to minimize the shared resource contention has been proposed by several authors [28, 52, 153]. Combining DVFS with application partitioning or scheduling has been widely studied [50, 80, 104, 112, 113, 152, 158, 167, 169, 174]. These works consider a multiprogrammed environment where applications appear as parallel tasks or can be partitioned into independent sub-tasks. Energy efficiency is then achieved by co-scheduling and power management. In Chapter 7, our first baseline power management policy, which separates resource-contending phases and performs DVFS to improve performance while saving energy, is inspired by these works.

Among the power management studies for CPU-GPU based heterogeneous systems that do not involve application manipulation, Singla et al. dynamically manage power and thermals using DVFS and varying the processor count and type [147]; Paul et al. coordinate power between the CPU and GPU for high performance computing applications [130]; and Pathania et al. vary the DVFS settings of the CPU and the GPU for 3D gaming workloads [127]. However, the benchmarks considered in these works do not utilize the CPU and the GPU simultaneously. Our second baseline power management policy, presented in Chapter 7, is inspired by these works.

With the advent of Heterogeneous System Architecture [12] introduced by the HSA Foundation [9], programmers have begun to utilize the CPU and the GPU concurrently to improve application performance [124, 154]. Power management for such applications has not been previously explored in detail. In

Chapter 7, we simulate real-world heterogeneous applications by overlapping SPEC and GPGPU benchmarks of different characteristics, and demonstrate significant energy savings with negligible performance impact with respect to state-of-the-art power management policies.

CHAPTER 3

ENERGY-EFFICIENT AUTOMATED MEETING ROOM ASSIGNMENT

3.1 Introduction

Recent research in the area of building automation has focused on making the HVAC systems more adaptive to changing conditions. Occupancy sensing and prediction [21, 47, 55, 76, 81, 95, 115, 122] involves the use of cameras, IR motion sensors, optical tripwires, badge readers, or custom presence detectors to track building occupants and react accordingly. For example, HVAC conditioning can be reduced in a currently unoccupied zone, or in one that is predicted to become imminently unoccupied.

As a complement to reactive approaches such as occupancy prediction, we envision future intelligent building systems taking acceptably benign *proactive* measures to affect occupant behavior in a way that improves building energy efficiency. If one considers the occupants as the building workload, reactive techniques such as occupancy prediction take action upon detected changes in the workload–*e.g.*, occupant movements–whereas proactive approaches attempt to suitably *shape* the workload, to cause the occupants to take relatively benign actions that are more energy-friendly.

One such way to affect the occupant behavior is by automatically assigning meetings to the rooms such that the overall HVAC energy consumption is minimum. The meeting room assignment problem is NP-complete [125] and is conventionally performed on a first-come, first-served basis. The typical "algorithm" matches the size of the meeting to the capacity of the available rooms.

However, given a complex schedule of meeting times and occupied rooms, there are many possible room assignments and great differences in their energy usage. The determination of a reasonably energy-efficient room assignment becomes a non-trivial exercise due to the many factors that impact the energy use of a meeting assignment.

In this chapter, we develop an abstract analytical energy model and capture the critical parameters impacting the meeting room scheduling energy savings. Using this model, we propose several meeting assignment algorithms, such as greedy and backtracking search, and a 0-1 Integer Linear Programming formulation. We show that our assignment algorithms guided by our energy model achieve global optimality for a large number of meeting schedules. We also show that backtracking search does not scale well for larger problems, while greedy search achieves high scalability with some loss in optimality. Our 0-1 ILP formulation achieves global optimality and is more scalable than backtracking search.

We further characterize how meeting room, meeting schedule, building, and outside weather parameters impact the potential energy savings and identify the conditions favorable for significant energy savings, and when it is not worthwhile. We validate the intuition derived from our model through EnergyPlus simulations and show that our model accurately estimates energy savings.

Table 3.1: Key variable definitions considered in this chapter.

Variables	Definitions
n	Number of rooms
m	Number of meetings
$R_j.cap$	Capacity of a j^{th} room
$M_i.st$	Start time of an i^{th} meeting
$M_i.et$	Start time of an i^{th} meeting
$M_i.size$	Size of an i^{th} meeting
P_a, j, t	Active power of room R_j at timestep t
$P_{sl,j,t}$	Sleep power of room R_j at timestep t
P	Power function
E_j	Energy of room R_j
$E_{a,j}$	Active energy of room R_j
$E_{sl,j}$	Sleep energy of room R_j
$E_{ov,j}$	Overhead energy of room R_j
r_j	Room size ratio of room R_j with respect to the smallest room
occ_j	Occupancy of room R_j
$occ_{j,A}$	Occupancy of R_j under assignment A
$\beta_{j,t}$	Ratio of sleep to active power for room R_j at timestep t
$\gamma_j(occ_j)$	Ratio of the active power of a partially occupied room R_j with respect to its full occupancy
η_t	Energy savings at timestep t
η	Net energy savings
τ	Time until the room temperature naturally drops to T_{lower} during a meeting gap
t_o	Offset time
$\mathbb{T}$	Total time over which energy is measured
T_s	Set temperature
T_r	Room temperature
$T_{r,a}$	Room temperature during active state
$T_{r,sl}$	Room temperature during sleep state
T_o	Outdoor temperature
T_{upper}	Upper set temperature
T_{lower}	Lower set temperature
$X_{i,j}$	Boolean variable that is *true* if meeting M_i is assigned to room R_j
$X_{i,j,t}$	Boolean variable that is *true* if meeting M_i is assigned to room R_j at timestep t
$Y_{A,j,t}$	Boolean variable that is *true* if room R_j is occupied at timestep t under assignment A
A_j	Set of meetings assigned to room R_j
$\mathbb{S}_j$	Number of effective meetings assigned in room R_j

3.2 Problem Statement

In this section, we formally introduce the energy-efficient meeting room assignment problem and prove that finding a feasible assignment is in general NP-complete.

3.2.1 Formulation

Given a set of meetings and available rooms, the overall objective of meeting room scheduling is to allocate rooms to the meetings such that the HVAC energy to condition the rooms is minimized while maintaining the desired set temperature when a room is occupied (*Active*). Between meetings when the room is unoccupied, the set temperature is lowered to a *Sleep* temperature[1].

Let R and M be the set of n rooms and m meetings, respectively. For an i^{th} meeting $M_i \in M$, $M_i.st$, $M_i.et$, $M_i.size$, and $M_i.room$ represent, respectively, the start time, end time, meeting size, and the room where M_i is assigned. Initially, $M_i.room = \phi$. For a j^{th} room $R_j \in R$, $R_j.cap$ denotes its capacity. The set A_j contains the meetings assigned to room R_j. The room assignment problem is formulated as an optimization problem.

$$\min_{A_1, \ldots, A_n} \sum_{j=1}^{n} E_j\left(A_j\right), \qquad \text{where } M_i \in A_j \text{ if } M_i.room = R_j \tag{3.1}$$

subject to:

- Timing

$$M_i.st < M_i.et \leq M_k.st < M_k.et \tag{3.2}$$

$$\forall j \text{ and } \forall i \neq k \text{ with } M_i.room = M_k.room = R_j$$

- Capacity

$$M_i.size \leq R_j.cap \qquad \forall M_i.room = R_j \tag{3.3}$$

[1]Since it takes time to heat a room, the transition from *Sleep* to *Active* set temperatures occurs some time before the start of the meeting. Similarly, since occupants may linger at the end of a meeting, the transition from *Active* to *Sleep* at the scheduled meeting end time is delayed.

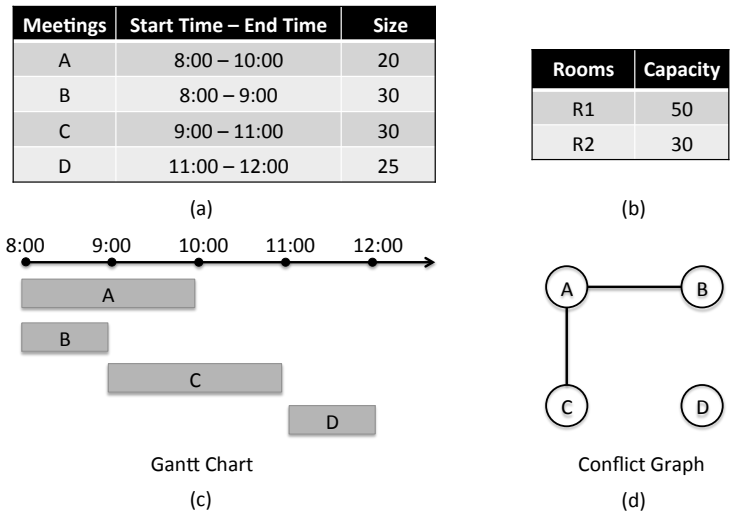


Figure 3.1: Example showing meeting scheduling as an edge coloring problem: (a) Schedule of meetings and their sizes; (b) Available meeting rooms and capacities; (c) Gantt chart; and (d) Conflict graph.

The objective is to determine a meeting assignment to rooms that minimizes the total HVAC energy over all meeting rooms while maintaining thermal comfort (set temperature objectives). The assigned meetings must be free from timing conflicts (meetings do not overlap within the same room) and capacity mismatches (an assigned meeting does not exceed the room capacity). The objective function is described below, where E_j denotes the energy of room R_j.

Figures 3.1 (a) and (b) show an example of a meeting scenario with four meetings and two rooms. Figures 3.1 (c) and (d) represent the timing conflicts between meetings through a Gantt chart and a conflict graph, respectively. Each node of the conflict graph represents a specific meeting and an edge between the two nodes indicates that the corresponding meetings overlap in time and they should not be assigned to the same room. Although this conflict graph is an interval graph [58] and polynomial-time graph coloring algorithms exist for such perfect graphs, we will show in the next section that the room capacity constraints render this seemingly tractable problem NP-complete.

3.2.2 Problem Complexity

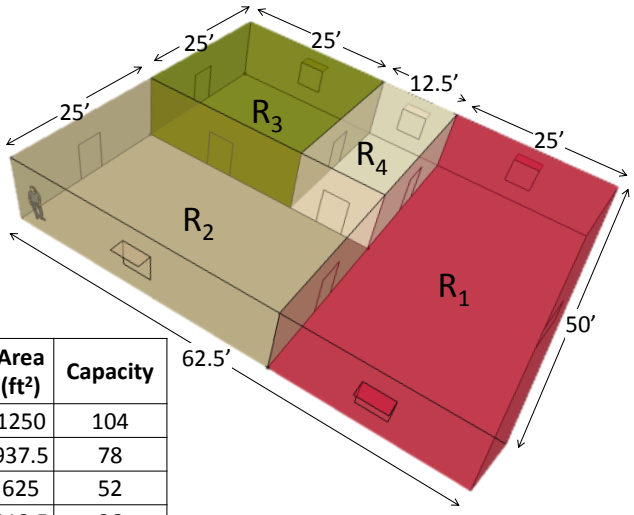
We formulate a list coloring problem on a set of intervals as the problem of determining the feasibility of our meeting room scheduling problem. The list coloring problem of intervals is stated as follows: Given a set of intervals I and a set of colors C, where each interval $i = (s_i, l_i, C_i) \in I$ is associated with a starting position s_i, a length l_i, and a list of colors $C_i \subseteq C$, find a coloring where the color of each interval i is taken from C_i and the intersecting intervals receive distinct colors.

The reduction process is the following: For each interval i, we construct a meeting M_i and associate it with an appropriate starting time and duration using s_i and l_i. For each color c, we construct a room R_j and associate it with a distinct positive capacity value $R_j.cap$. Hence, each color list C_i corresponds to a list of rooms $\{R_{j_1}, \ldots, R_{j_n}\}$. Then we can assign the size of a meeting M_i with $M_i.size = min\{R_{j_1}.cap, \ldots, R_{j_n}.cap\}$.

A feasible assignment of our meeting room problem directly corresponds to a legal coloring for the original interval set. Since the list-coloring problem for interval graphs is in general NP-complete [27], and we can compute the interval representation of a interval graph in linear time [72], the feasible meeting room scheduling problem is also NP-complete.

3.3 Building Power Characterization

The analytical model that we develop is based on understanding how various factors impact the meeting room assignment energy savings. In this section,



Rooms	Area (ft^2)	Capacity
R_1	1250	104
R_2	937.5	78
R_3	625	52
R_4	312.5	26

Figure 3.2: Layout of the simulated building.

we characterize the building power behavior, and identify the key factors that affect the energy savings. We first describe our modeling infrastructure and then present results from the characterization experiment.

3.3.1 Modeling Infrastructure

Modeling and energy characterization is performed using the Department of Energy's building energy simulation software EnergyPlus version 8.2 [6]. The layout is designed using the Google Sketchup Tool [11]. The construction material for building surface and fenestration are exported from an existing large office template, and lighting and electrical equipment usage are assumed to be on throughout the day to minimize its effect on HVAC energy. Each room maps to a unique thermal zone, and is controlled by an individual zone-level thermostat. The HVAC control system uses the default *IdealLoadsAirSystem*. We assume negligible building infiltration and enable Demand Controlled Ventilation to maintain occupancy comfort with minimum energy expense. To meet

the thermal comfort and indoor air quality, we maintain a minimum outdoor airflow of 5 cfm per person and 0.06 cfm per square feet of room area as per ASHRAE's 2010 ventilation standards [15].

The layout of the simulated building is shown in Figure 3.2. The building contains four rooms of different sizes and capacities. We eliminate other spaces (offices, hallways, *etc*.) from the layout in order to focus on the meeting-room energy savings. Room capacities are computed by assuming 12 square feet of area per person for a theatre style room [103]. We evaluate heating energy during the winter in Minneapolis (from January 28^{th} to February 1^{st}).

3.3.2 HVAC Power Characterization

In this section, we discuss the results of several experiments that inform the construction of our energy model. Figure 3.3 shows the general power behavior of a meeting room operating using *Active* and *Sleep* setpoint temperatures. When a room is occupied (*Active*), the heating setpoint is increased to T_{upper} (21°C in our experiments). Between meetings when a room is unoccupied, or when a room is no longer needed for the day, the heating setpoint is lowered to the *Sleep* setpoint, T_{lower} (15.6°C in our experiments). A room in *Sleep* begins transitioning to *Active* an offset time, t_o, prior to the start of a meeting so that the building occupants are comfortable when the meeting begins[2]. Similarly, when there is no subsequent meeting, the setpoint is lowered t_o minutes after the conclusion of the meeting in case occupants linger in the room.

[2]We experimentally verified that t_o = 15 minutes is sufficient time to condition the room to a comfortable thermal state.

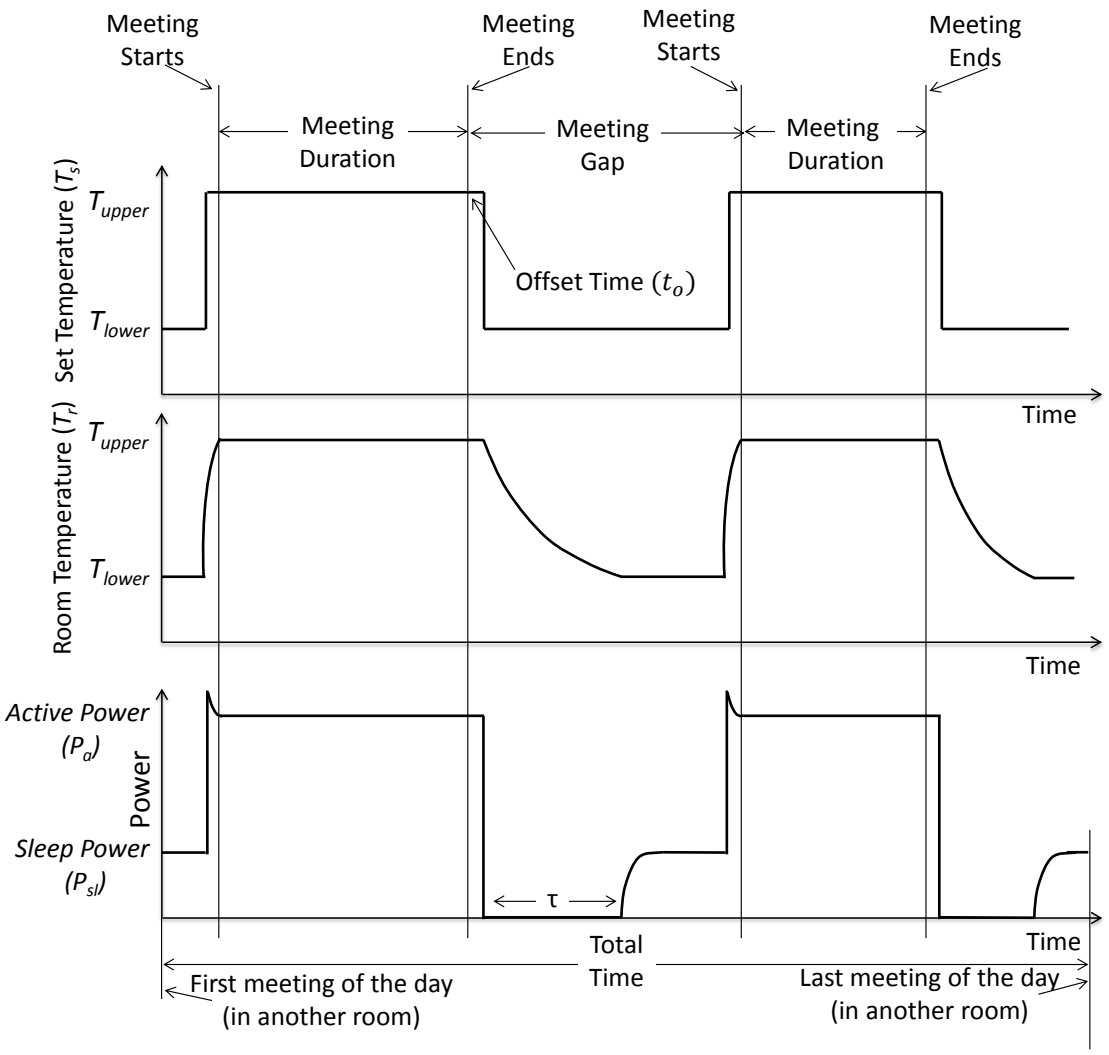


Figure 3.3: Depiction of the HVAC power behavior for meeting rooms with *Active* and *Sleep* setpoint temperatures.

When the setpoint is increased from T_{lower} to T_{upper}, the room temperature takes t_o to reach the setpoint. The power, however, shows a spike before settling down to a steady value. This is because the HVAC control uses room temperature as feedback. On changing the setpoint, the controller observes a large error in the room temperature and the setpoint, and thus expends high energy to mitigate this error. As the room temperature reaches the setpoint, the error reduces and the power settles to a non-zero value. The power is non-zero because it maintains the room at T_{upper} while bringing in and conditioning outside air to

maintain acceptable indoor air quality. After t_o minutes have passed from the scheduled end of the meeting, the room is assumed to be empty, and thus the heating setpoint is reduced to T_{lower}. Thus, the HVAC system shuts down so that the room temperature naturally decays towards T_{lower}, with zero HVAC power. This time of zero power, τ, depends on outdoor weather, how long the room has been warmed up (thermal state), and building heat loss characteristics. After time τ, the HVAC controller turns on to maintain the room temperature at T_{lower}.

The intuition derived from this characterization permits constructing an analytical energy model that relates to the meeting and room specifications, as well as building and environmental conditions.

Active Power

Active power (P_a) is the HVAC power of a zone when it is occupied and set to the T_{upper} setpoint. We evaluated a number of factors that affect the active power.

Room Capacity: We first evaluated the active power of each of the four meeting rooms in order to understand the relationship between active power and room capacity and the effect of outside temperature. In our experiments, the HVAC system maintains the T_{upper} setpoint temperature throughout the day over a five day period with full occupancy. Figure 3.4 shows the heating power per unit room area for each of the rooms during different periods of the day. The ratio for the different rooms is very close, which indicates a linear relationship between room area and active heating power. We observe that room R_4 has a slightly

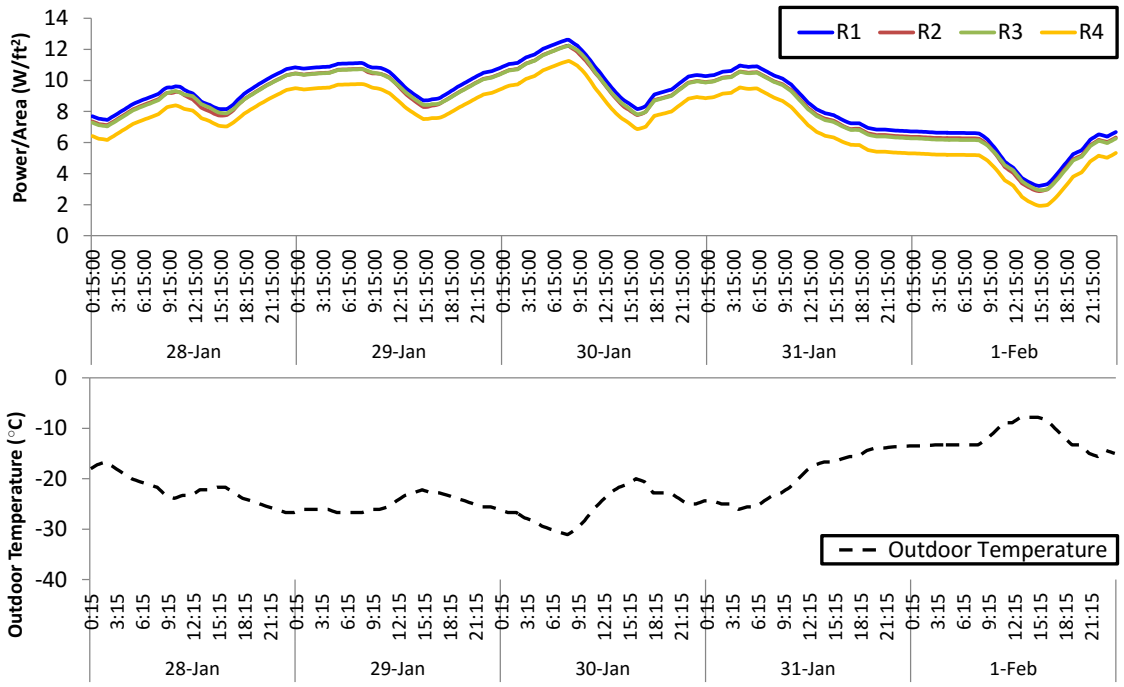


Figure 3.4: Active power per unit area of the different rooms, and outdoor air temperature, over a five day period.

lower ratio, which we experimentally determined was due to R_4 being a small room that experiences lateral heat transfer from neighboring rooms because of our particular building layout, where R_4 has three walls adjacent to other rooms. This results in R_4 expending less W/m^2 than the other rooms. From Figure 3.4, as expected, the active power increases with colder weather, and vice-versa. This is because more heating energy is needed to warm up the colder outdoor air brought in to maintain the indoor air quality. Note, however, that all the rooms respond similarly to the changes in outdoor temperature.

Room Occupancy: With Demand Controlled Ventilation, more outdoor air is brought in during periods of high occupancy in order to maintain acceptable indoor air quality. As a result, higher occupancy requires more heating power to heat the outdoor air. We explored the effect of partial occupancy on the heating power as a function of room size and outdoor temperature. We define γ as

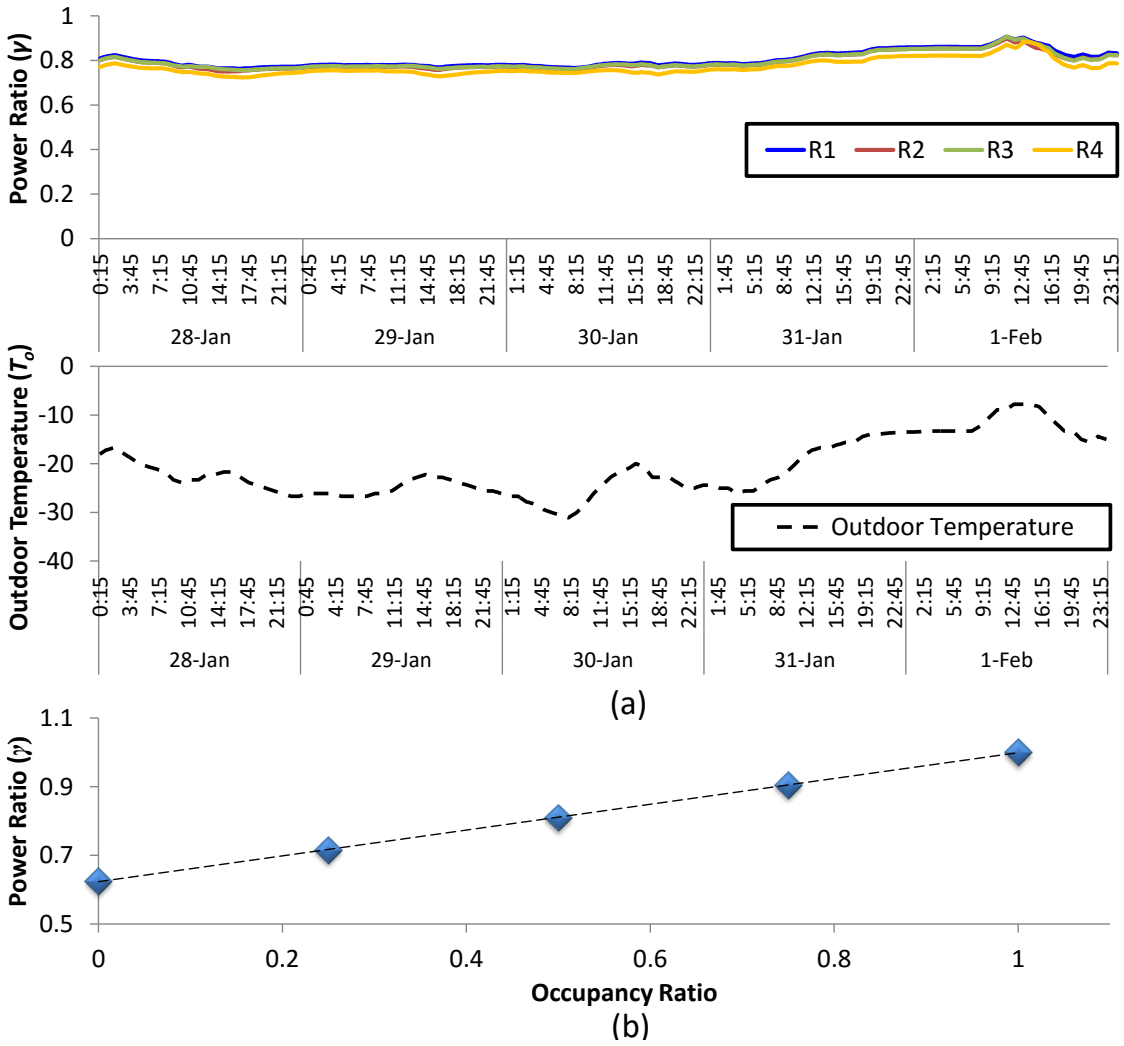


Figure 3.5: (a) Effect of room size and outdoor temperature on γ for half occupancy. (b) Effect of partial occupancy on active power.

the ratio of the active power of a partially occupied room to a fully occupied one. Figure 3.5(a) shows γ at half occupancy for the different room sizes. As expected, γ is less than one, but the effect of both room size and outside temperature is small. Figure 3.5(b) further shows that active power per unit area increases linearly with occupancy, since more outside air is required to maintain acceptable indoor air quality.

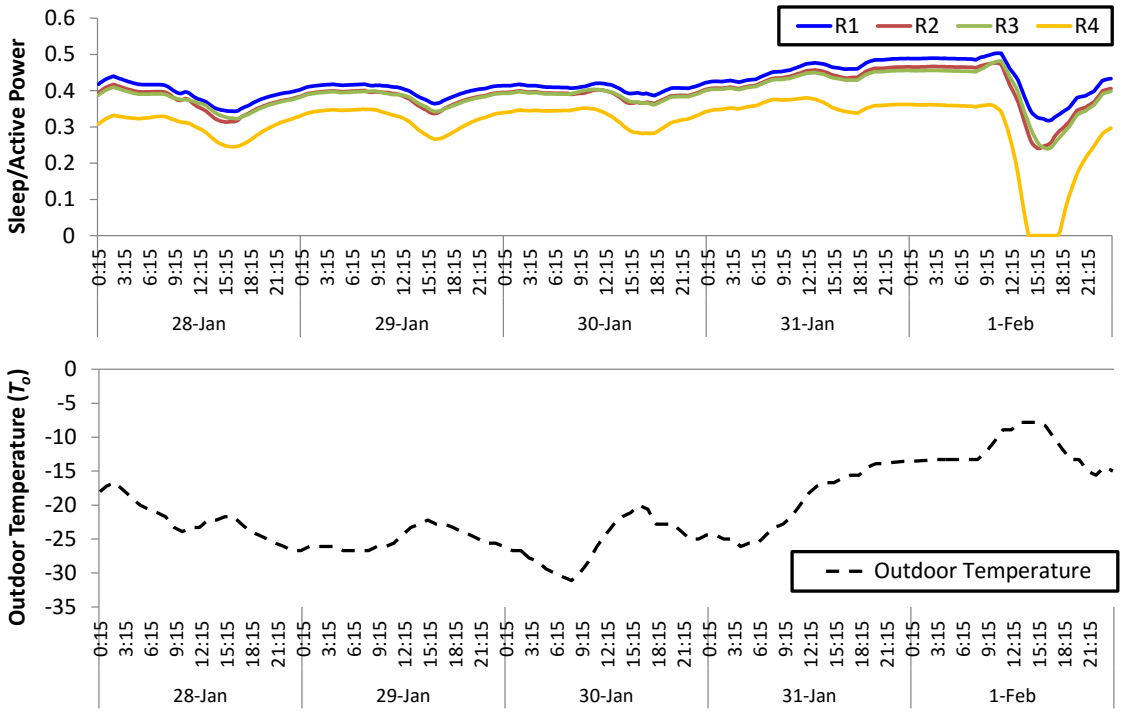


Figure 3.6: Effect of room size and outdoor temperature on the sleep to active power ratio (β).

Sleep Power

Sleep power is the HVAC power to maintain the room temperature at T_{lower} during periods of no occupancy after the decay period τ. During this period, the desired set temperature is lowered as is the outdoor airflow, which can significantly reduce the heating power.

We define β as the ratio of the sleep to active power, and we explore the effect of room size and outdoor temperature on this parameter. Figure 3.6 shows the value of β for the four rooms throughout the five day period. Here, we model the *Sleep* power through the five day period (with the set temperature at T_{lower} throughout) and the *Active* power (with the set temperature at T_{upper}), and then take the ratio. As expected, β is less than one since sleep power is smaller than active power. Moreover, the value of β is inversely proportional to the outside

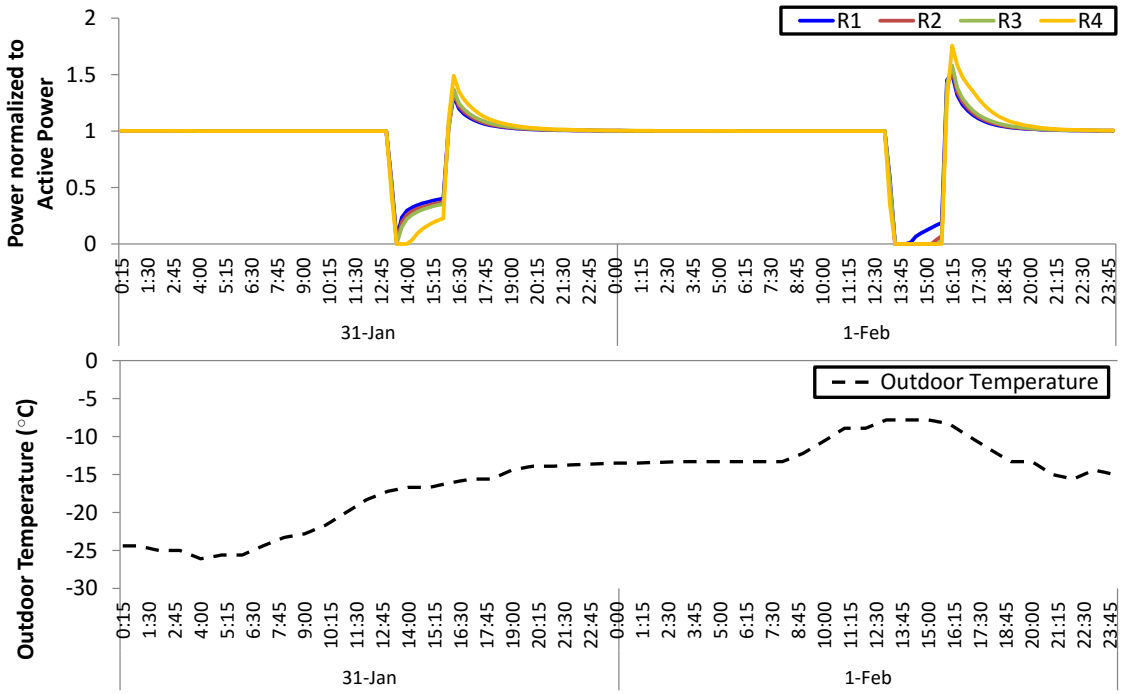


Figure 3.7: Sleep-to-Active power ratio during meeting gaps.

temperature. The reason for this behavior is that the HVAC power is roughly proportional to the difference in the room and outside air temperatures. Thus, as the outside temperature increases, the change has a greater effect on reducing required *Sleep* power compared to *Active* power. Furthermore, β is relatively independent of room size, except for R_4. This is because of lateral heat transfer from shared walls.

We now investigate the effect of room size and outdoor temperature on τ, the time for the room temperature to decay to T_{lower} when the HVAC system is turned off. Figure 3.7 shows the power relative to the *Active* power over a two day period (blown up for clarity) and room sizes when meetings are periodically assigned for a time period followed by a time gap with no assigned meetings. Our results for the full five day period show that τ varies widely with outdoor temperature, from 5 minutes to an hour. It also varies across different

rooms. A longer τ results in less sleep power, since the HVAC system can be shut down for longer periods during meeting gaps. Furthermore, we observe a power spike when the setpoint is increased to T_{upper}, due to the large temperature error observed by the HVAC controller. While our model currently ignores this power spike, we show later that our accuracy is still very high.

3.4 Energy Model

In this section, we present an energy model derived from the building power characterization. The model estimates the *Active* energy when a meeting is scheduled, the *Sleep* energy when no meeting is scheduled, and the *Overhead* energy during transitions. Before we present the individual models for each of these energy phases, we review some of the key variables. The complete list can be found in Table 3.1.

3.4.1 Key Variable Definitions

Let $X_{i,j,t}$ be a Boolean variable that indicates that room R_j is assigned to meeting M_i at timestep t. When $X_{i,j,t} = true$, it indicates meeting M_i is assigned to room R_j, *i.e.*, $M_i.room = R_j$. At timestep t, variable $\beta_{j,t}$ is the ratio of sleep, and active power of R_j when it is fully occupied, i.e. $occ_j = R_j.cap$. It is a function of room temperature (T_r), and the outdoor air temperature (T_o), and is formally defined in Equation 3.4. Note that $\beta_{j,t}$ is zero for a decay time of τ.

$$\beta_{j,t} \triangleq \begin{cases} \frac{P_{sl,j,t}}{P_{a,j,t}} = \frac{P(T_{lower},T_{r,sl},T_o,0)}{P(T_{upper},T_{r,a},T_o,R_j.cap)}, & \text{if } T_{r,sl} \leq T_{lower} \\ 0, & \text{otherwise} \end{cases} \tag{3.4}$$

We also define γ_j in Equation 3.5 as the ratio of the active power when room R_j is partially occupied with occupancy occ_j compared to the active power when it is fully occupied. As γ_j and occ_j are linearly related, we employ a linear regression model and use the slope and intercept from Figure 3.5(b).

$$\gamma_j(occ_j) \triangleq \frac{P(T_{upper}, T_{r,j}, T_o, occ_j)}{P(T_{upper}, T_{r,j}, T_o, R_j.cap)} \tag{3.5}$$

3.4.2 Active Energy

From the building power characterization, the active power at timestep t is a function of room capacity and occupancy. At full occupancy, the active power is proportional to the room capacity. For less than full occupancy, the active power is expressed as Equation 3.6.

$$P_{a,j,t} \propto \gamma_j(occ_j) \times R_j.cap \tag{3.6}$$

The active energy of a room j is the sum of its active power over the total time $\mathbb{T}$, and is proportional to the capacity of R_j times the total duration of all the meetings assigned to it. Here, $X_{i,j}$ is a Boolean variable, which is *true* if meeting M_i is assigned to room R_j.

$$\begin{aligned} E_{a,j} &= \sum_{t=0}^{\mathbb{T}} P_{a,j,t} \\ &\propto R_j.cap \times \sum_{i=1}^{m} X_{i,j} \times \gamma_j(M_i.size) \times (M_i.et - M_i.st) \end{aligned} \tag{3.7}$$

3.4.3 Sleep Energy

Sleep power is expressed as β times the active power of a room when fully occupied. β determines the degree of energy savings when conditioned at a lower set temperature, and depends on room temperature and outside weather conditions. Equation 3.8 shows this relationship.

$$\begin{aligned} P_{sl,j,t} &= \beta_{j,t} \times P_{a,j,t} \\ &\propto \beta_{j,t} \times R_j.cap \end{aligned} \tag{3.8}$$

Equation 3.9 shows the sleep energy formulation.

$$\begin{aligned} E_{sl,j} &= \sum_{t=0}^{\mathbb{T}} P_{sl,j,t} \\ &\propto R_j.cap \times \sum_{t=0}^{\mathbb{T}} \overline{X_{i,j,t}} \times \beta_{j,t} \end{aligned} \tag{3.9}$$

3.4.4 Overhead Energy

Every transition between *Active* and *Sleep* incurs *Offset* overhead to ensure thermal comfort requirements. Before a meeting starts, the overhead is incurred to condition the room before occupants arrive. At its conclusion, the overhead is incurred to keep the room conditioned for lingering occupants. To model the overheads, we compute the number of switching activities and multiply by the energy overheads. During these periods, the room is assumed unoccupied. Equation 3.10 formulates the energy from this overhead.

$$E_{ov,j} \propto \mathbb{S}_j \times \gamma_j(0) \times R_j.cap \times (2 \times t_o) \tag{3.10}$$

Here, $\mathbb{S}_j$ represents the number of *effective meetings* assigned to room R_j. If two meetings are scheduled back to back with no gap, these two meetings can be simply considered as one longer effective meeting. $\mathbb{S}_j$ is computed by summing the total meetings assigned to room R_j minus the factor $W_{pq,j}$ as shown below.

$$\mathbb{S}_j = \sum_{i=1}^{m} (X_{i,j} - W_{pq,j}) \tag{3.11}$$

$W_{pq,j}$ accounts for the meeting pairs with no gaps in between. This is formulated as Equation 3.12.

$$W_{pq,j} = \begin{cases} 0 & \text{if } p.et = q.st \\ X_{p,j} \wedge X_{q,j} & \text{otherwise} \end{cases} \tag{3.12}$$

3.4.5 Room Energy

The energy of a room R_j is the sum of active energy ($E_{a,j}$), sleep energy ($E_{sl,j}$) and the overhead energy ($E_{ov,j}$) , as shown in Equation 3.13.

$$E_j = E_{a,j} + E_{sl,j} + E_{ov,j} \tag{3.13}$$

3.5 Meeting Room Assignment Algorithms

In this section, we use our proposed energy model to solve the meeting room assignment problem. We develop search algorithms and a 0-1 Integer Linear Programming (*0-1 ILP*) formulation to find the most energy optimum meeting assignment. Both methods use our energy model as a proxy to represent the energy consumption when meetings are assigned to rooms.

3.5.1 Search Algorithms

We first present algorithms with two search methodologies - *greedy* and *backtracking*. Both use meeting times, the number of attendees, and room capacities to build a meeting room search tree. These algorithms search for a feasible assignment that gives the least energy consumption according to our energy model.

Greedy Search

The Greedy Search (*Greedy*) algorithm greedily searches for the assignment with the lowest energy according to the energy model. This algorithm does not backtrack, and therefore has polynomial time complexity. It may produce a suboptimal solution, or may not find a solution altogether.

The meetings are first assigned in the order of their room choices based on the capacity constraint, from fewest choices to most choices. When multiple meetings have an equal number of room choices, meetings are processed in the order of increasing start time (earlier meetings are handled first).

For the non-conflicting situation with only one meeting between any two time stamps, *Greedy* explores all the room choices that meet the constraints, and assigns that room which gives the smallest energy value.

For conflicting meetings starting at the same time, there are two cases to consider. In the first case, the conflicting meetings have different room choices that minimize the energy value, in which case each conflicting meeting is assigned to its lowest-energy room. In the second case, multiple meetings have the same

lowest-cost room. For m conflicting meetings, finding a globally optimum assignment requires searching among O(m!) options, which makes it exponential.

We developed an effective greedy strategy for this situation in order to reduce the computational complexity to polynomial time. For each conflicting meeting, we determine the cost difference between the lowest cost and second lowest cost room choices. The meeting with the highest cost difference is given the lowest cost room. For the remaining meetings, we iterate with the second lowest cost option, and continue until all conflicting meetings are assigned a room.

Backtracking with Reduced Search

The Backtracking with Reduced Search (*BT-reduced*) algorithm performs a depth-first search of the meeting-room search tree for a feasible solution, and then backtracks to other feasible solutions to find the energy optimum meeting assignment. *BT-reduced* also performs branch-and-bound to reduce the branching factor of the search tree by avoiding those depth searches whose sub-optimality can be determined *a priori*. During the depth-first search of various room options for meeting M_i in the i^{th} recursion, the algorithm evaluates the estimate of the optimality of the generated assignment through the energy model. Whenever the algorithm reaches a leaf node ($i = m$), it has found a feasible solution, and the minimum energy value (along with its corresponding assignment) is updated. The $(i+1)^{th}$ recursive step is called only if the minimum energy value is greater than that of the current optimal assignment. Otherwise, the search tree is pruned at this point.

3.5.2 0-1 Integer Linear Programming

In this section, we use the energy model and formulate the meeting-room assignment problem as a 0-1 ILP formulation. Formulating the assignment problem as an ILP motivates us to use fast solvers, such as CPLEX [17], to achieve higher scalability towards larger problems compared to search-based backtracking algorithms.

Objective Function

Recall from Section 3.4.5 that room energy comprises active, sleep and overhead energy components. Equation 3.14 shows the objective function, where we minimize the total HVAC energy across all n rooms.

$$\min \quad \sum_{j=1}^{n} (E_{a,j} + E_{sl,j} + E_{ov,j}) \tag{3.14}$$

Constraints

Consistency Constraint: To ensure all the meetings are assigned and each meeting is assigned to exactly one room, the constraints are laid out as per Equation 3.15.

$$\sum_{j=1}^{n} X_{i,j} = 1 \tag{3.15}$$

$$\forall 1 \leq i \leq m$$

Capacity Constraint: The size of a meeting cannot exceed the capacity of a room, which leads to the following constraint:

$$X_{i,j} = 0 \qquad \text{if} \quad M_i.size > R_j.cap \tag{3.16}$$

In order to reduce the solver's execution time, we do not include variable $X_{i,j}$ in our 0-1 ILP formulation for the cases where M_i cannot be assigned to R_j.

Timing Constraint: We add a set of timing constraints for conflicting meeting pairs. For the example of Figure 3.1, we add the constraints of Equation 3.17.

$$\begin{aligned} X_{A,j} + X_{B,j} &\leq 1 \\ X_{A,j} + X_{C,j} &\leq 1 \end{aligned} \qquad (3.17)$$

$$\forall 1 \leq j \leq n$$

Note that the $\leq$ relation is used because it is acceptable to have both $X_{A,j}$ and $X_{B,j}$ to be *false*, which indicates that they are assigned to different rooms other than room R_j.

Meeting Gaps: Recall from Equation 3.12 that $W_{pq,r}$ involves a Boolean *AND* operation between $X_{p,r}$ and $X_{q,r}$. This makes the model quadratic. To formulate this as an ILP, we linearize Equation 3.12 by adjoining Fortet inequalities [59], as shown in Equation 3.18.

$$\begin{aligned} W_{pq,j} &\leq X_{p,j} \\ W_{pq,j} &\leq X_{q,j} \\ W_{pq,j} &\geq X_{p,j} + X_{q,j} - 1 \\ 0 \leq W_{pq,j} &\leq 1 \end{aligned} \qquad (3.18)$$

The above linearization ensures that $W_{pq,j}$ is always *true* if $X_{p,j}$ and $X_{q,j}$ are *true*. However, since $W_{pq,j}$ factors in the gap between meeting M_p and M_q, ideally it should be *true* if and only if there are no other meetings scheduled between M_p and M_q (provided both M_p and M_q are assigned to the same room R_j). If they are

assigned to different rooms, $W_{pq,j}$ should be *false*. For this property to hold, we introduce another variable $V_{pq,j}$. We express the situation with t meetings that happen in between M_p and M_q in the increasing order of time, say z_1, z_2 ... z_t, using the constraints shown in Equation 3.19.

$$\begin{aligned} V_{pq,j} + X_{z_1,j} &\leq 1 \\ V_{pq,j} + X_{z_2,j} &\leq 1 \\ &\vdots \\ V_{pq,j} + X_{z_t,j} &\leq 1 \end{aligned} \tag{3.19}$$

Equation 3.19 ensures if any one meeting is scheduled between M_p and M_q, $V_{pq,j}$ is *false*. We also capture the condition when $W_{pq,j}$ is *false* and $V_{pq,j}$ is forced to zero:

$$\begin{aligned} V_{pq,j} &\leq W_{pq,j} \\ V_{pq,j} + X_{z_1,j} + X_{z_2,j} ... + X_{z_t,j} &\geq W_{pq,j} \end{aligned} \tag{3.20}$$

However, with this approach, the total number of W and V variables for each x and y pair grows quadratically, as per $\binom{mn}{2}$. In order to reduce the number of variables, we eliminate those pairs that overlap in time since they can never be assigned to the same room. Furthermore, while considering the in-between meetings, we avoid those that fail the capacity constraint. In this way, we reduce the number of V and W variables for selected meeting pairs in order to achieve faster solver performance.

3.6 Energy Savings

In this section, we describe the calculation of energy savings using the energy model. We first show a simple example, and then the general case. We also show how various model parameters and meeting room situations impact the potential energy savings of smart meeting assignment.

3.6.1 An Illustrative Example

Using the building layout of Figure 3.2 with $n = 4$ rooms, we consider assignments A and B, with B as the baseline. If $E_j|_A$ represents the room energy of R_j under assignment A, and $E_j|_B$ for assignment B, the energy savings η_t at timestep t is given by

$$\eta_t \triangleq 1 - \frac{\sum_{j=1}^{n} E_j|_A}{\sum_{j=1}^{n} E_j|_B} \tag{3.21}$$

Assume in this example that for a given timestep t, assignment A keeps room R_4 occupied with occupancy $occ_{4,A}$ and all other rooms unoccupied. Baseline B maintains R_1 with occupancy $occ_{1,B}$ while keeping the rest of the rooms unoccupied. In this case, $occ_{1,B} = occ_{4,A}$ and η_t is expressed as:

$$\begin{aligned}\eta_t &= 1 - \frac{E_{1,sl} + E_{2,sl} + E_{3,sl} + E_{4,a}(occ_{4,A})}{E_{1,a}(occ_{1,B}) + E_{2,sl} + E_{3,sl} + E_{4,sl}} \\ &= 1 - \frac{\beta_{1,t}E_{1,a} + \beta_{2,t}E_{2,a} + \beta_{3,t}E_{3,a} + \gamma_4(occ_{4,A})E_{4,a}}{\gamma_1(occ_{1,B})E_{1,a} + \beta_{2,t}E_{2,a} + \beta_{3,t}E_{3,a} + \beta_{4,t}E_{4,a}}\end{aligned} \tag{3.22}$$

Also, since the active energy at full occupancy is proportional to the room capacity, we can write

$$\eta_t = 1 - \frac{\beta_{1,t}R_1.cap + \beta_{2,t}R_2.cap + \beta_{3,t}R_3.cap + \gamma_4(occ_{4,A})R_4.cap}{\gamma_1(occ_{1,B})R_1.cap + \beta_{2,t}R_2.cap + \beta_{3,t}R_3.cap + \beta_{4,t}R_4.cap} \tag{3.23}$$

Dividing Equation 3.23 with the smallest room capacity $R_4.cap$ in both numerator and denominator, we get

$$\eta_t = 1 - \frac{\beta_{1,t}r_1 + \beta_{2,t}r_2 + \beta_{3,t}r_3 + \gamma_4(occ_{4,A})r_4}{\gamma_1(occ_{1,B})r_1 + \beta_{2,t}r_2 + \beta_{3,t}r_3 + \beta_{4,t}r_4} \tag{3.24}$$

where r_j is the room capacity ratio of R_j relative to a smallest room capacity. This formulation permits the use of relative rather than absolute room size as energy parameters when applying the model to new scenarios.

3.6.2 General Case

Equation 3.25 shows the energy savings model of a generic room occupancy situation for assignments A and B. Here, variable $Y_{j,A,t}$ is a Boolean variable, which is *true* if room R_j is occupied with occupancy $occ_{j,A}$ under assignment A at timestep t. Energy savings η_t at timestep t is given by

$$\eta_t = 1 - \frac{\sum_{j=1}^{n}\left[\beta_{j,t}r_j\overline{Y_{A,j,t}} + \gamma_j(occ_{j,A})r_jY_{A,j,t}\right]}{\sum_{j=1}^{n}\left[\beta_{j,t}r_j\overline{Y_{B,j,t}} + \gamma_j(occ_{j,B})r_jY_{B,j,t}\right]} \tag{3.25}$$

The net energy savings, η, over total time $\mathbb{T}$ is given by

$$\eta = 1 - \frac{\sum_{t=1}^{\mathbb{T}}\left[\sum_{j=1}^{n}\left\{\beta_{j,t}r_j\overline{Y_{A,j,t}} + \gamma_j(occ_{j,A})r_jY_{A,j,t}\right\}\right]}{\sum_{t=1}^{\mathbb{T}}\left[\sum_{j=1}^{n}\left\{\beta_{j,t}r_j\overline{Y_{B,j,t}} + \gamma_j(occ_{j,B})r_jY_{B,j,t}\right\}\right]} \tag{3.26}$$

Therefore, apart from the meeting assignment vector Y, energy savings η_t at any timestep t depends on the sleep to active power ratio (β), the room capacity ratio

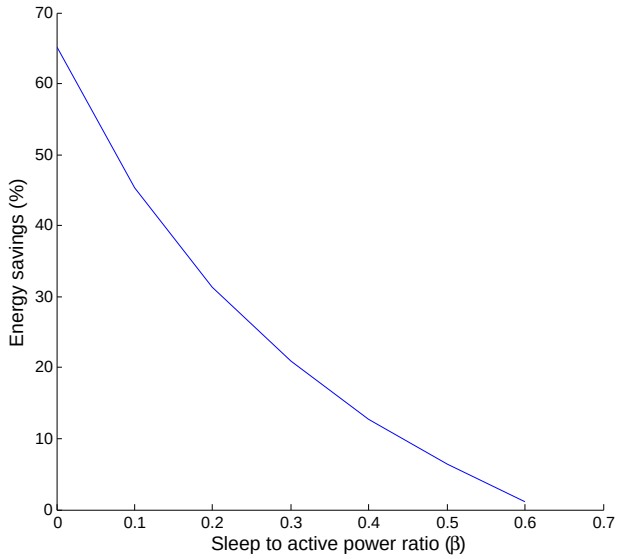


Figure 3.8: Effect of β on meeting room energy savings.

(r), and the number of rooms (n). We next illustrate how these factors affect the energy savings.

3.6.3 Factors Affecting Potential Energy Savings

We now show how varying the model parameters and meeting room situations impact the potential energy savings of smart meeting assignment. Specifically, we examine the sleep-to-active power ratio, the number of scheduled meetings, scheduling flexibility, and room size ratio.

Sleep-to-Active Power Ratio

We consider the building layout of Figure 3.2 and examine one meeting whose size equals that of the smallest room R_4. To understand the potential energy savings achievable with smart meeting assignment, we assume that the baseline

assigns the meeting to the largest room R_1, while the smart policy assigns it to the smallest room R_4. The unoccupied rooms in both cases are in the *Sleep* state.

Figure 3.8 shows that the energy savings increases as β decreases. For smaller β, the sleep power is relatively small compared to the active power, which yields higher savings from the use of the *Sleep* state for the three unoccupied rooms. Since the value of β depends on a number of factors, it serves as a proxy for various conditions. As shown earlier, more extreme weather conditions result in higher β; thus, in these conditions, a lower relative energy savings would be expected compared to more moderate times of the year[3]. Moreover, room characteristics and building layout may significant impact the value of β, and thus the expected savings.

Number of Scheduled Meetings

For a given number of rooms, increasing the number of meetings reduces the number that are unoccupied and in *Sleep* mode. This impacts the potential energy savings as shown in Figure 3.9. Here, we assume the same rooms R_1 to R_4 and that each scheduled meeting is the same capacity as the smallest room (R_4). The baseline algorithm assigns rooms in descending order of capacity (R_1 first) while the smart algorithm works in ascending order (R_4 first). As expected, the largest savings is achieved with a single meeting, since the smart algorithm puts the largest room in *Sleep* while the baseline chooses the smallest room. With all four rooms occupied, there is no benefit since all rooms are *Active* and have the same occupancy. This demonstrates that offices that frequently use all or most of their rooms at the same time would expect lesser savings that those that si-

[3]Note that our model only predicts *relative* energy savings. Depending on the conditions, the differences in *absolute* savings between different times of the year may be lower.

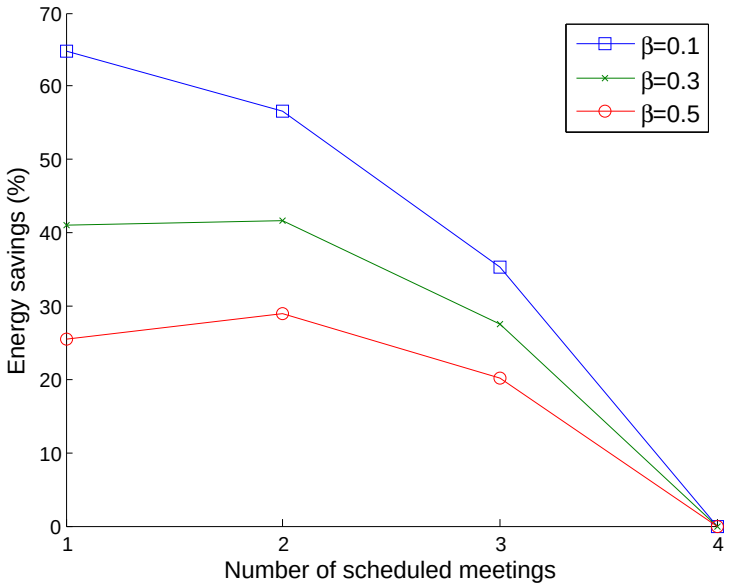


Figure 3.9: Effect of number of scheduled meetings on meeting room energy savings.

multaneously use fewer rooms.

The effect is more pronounced for smaller values of β where the potential savings while in *Sleep* mode is larger. Note how the energy savings increases slightly when moving from one to two meetings for the two larger values of β. This occurs because the energy difference between the two assignment policies reaches its maximum for two scheduled meetings, since no rooms are maintained at the same condition in both policies.

Scheduling Flexibility

To understand the effect of scheduling flexibility on potential energy savings, we consider a single meeting whose size is increased from the capacity of the smallest room (R_4) to the largest (R_1) and assume the same baseline and smart algorithms as before. As shown in Figure 3.10, as the size of the meeting increases,

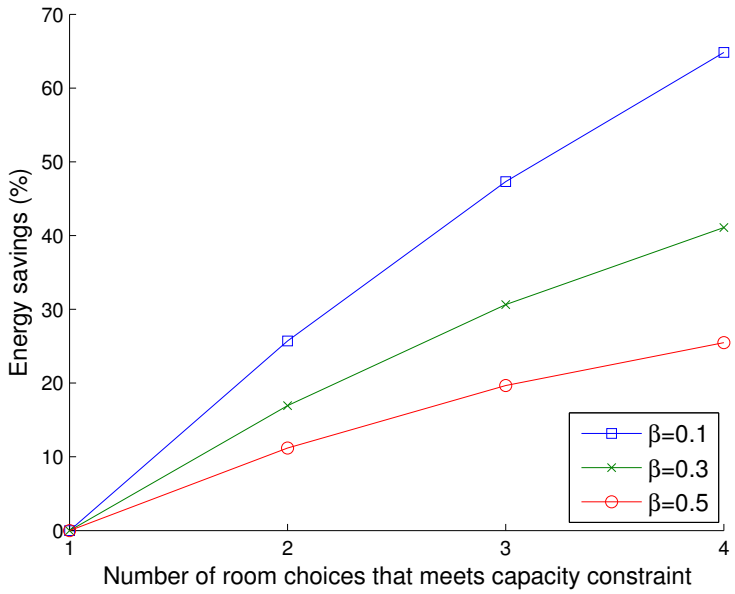


Figure 3.10: Effect of scheduling flexibility on meeting room energy savings.

there are fewer choices (less flexibility) and the difference between the baseline and smart policies diminishes. Thus, if larger meetings tend to be scheduled over smaller ones, then the expected savings from smart scheduling would be expected to be smaller. Note also the effect of β: the relative improvement with maximum flexibility can vary by a factor of three.

Unused Rooms and Room Size Ratio

Lastly, we examine the effect of the number of unused rooms and their relative size ratios on the energy savings. We assume n rooms, of which $n-1$ are of equal size, and the last room is r times larger. For this experiment, we assume $\beta = 0.5$. We consider a single meeting where the baseline uses the largest room and the smart algorithm one of the other smaller rooms.

As shown in Figure 3.11, as n increases, the proportional amount of *Sleep* energy increases, which results in lower overall energy savings. On the other

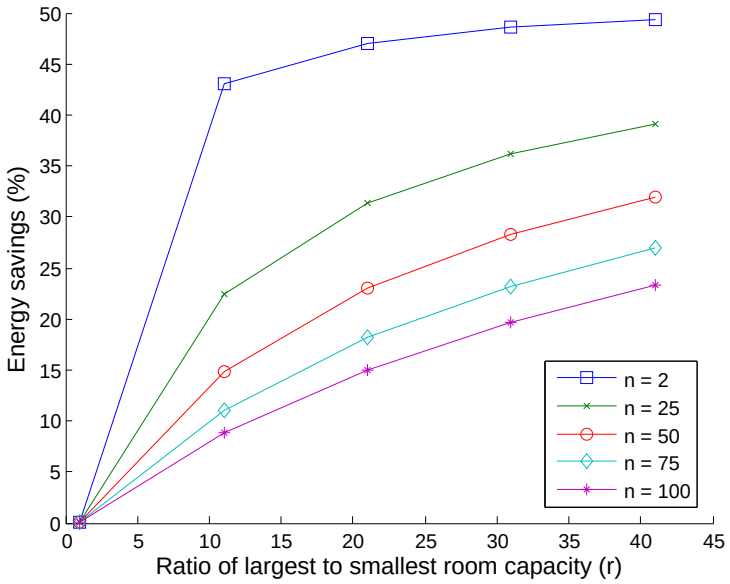


Figure 3.11: Effect of n and r on meeting room energy savings.

hand, as r increases so does the energy savings, since the differences in *Active* energy increases.

3.7 Evaluation Methodology

In order to examine the energy savings and runtime performance of different assignment algorithms, Figure 3.12 presents the meeting benchmarks we study in this chapter. The first five benchmarks are synthetically designed to test special cases. The last two benchmarks are randomly generated meeting schedules. The benchmarks are represented by graphs, with nodes representing time stamps, edges indicating meetings, and the number in brackets indicating the meeting size. Meetings are scheduled between 8am and 6pm with a minimum duration of one hour. We also annotate each meeting benchmark with its average number of scheduled meetings and scheduling flexibility. The formulas used to compute these two metrics are shown in Equation 3.27.

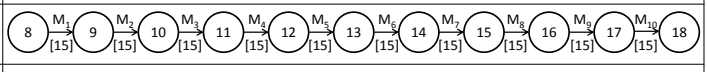
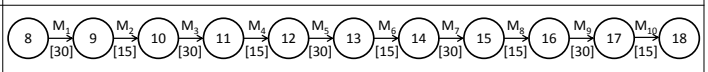
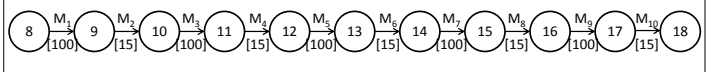
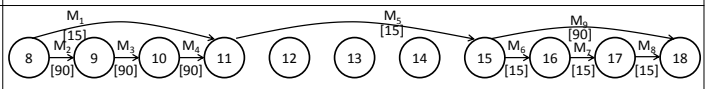
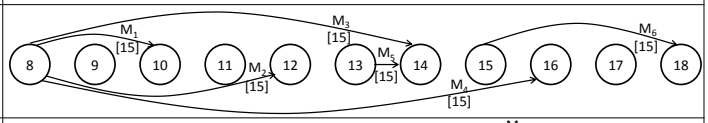
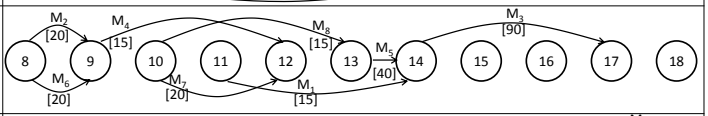
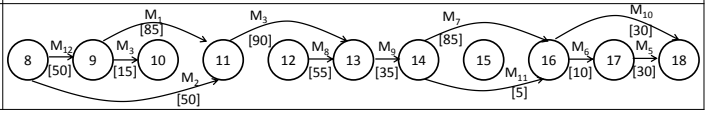
Meeting Benchmarks		Avg. Sched. Mtgs.	Avg. Sched. Flex.	Graphical Representation of the Meetings
SYNTHETIC	10 continuous meetings with size 15 (10c_15)	Low [1]	High [4]	
	10 interval meetings with size 15 and 30 (10i_15_30)	Low [1]	Medium [3.5]	
	10 interval meetings with size 15 and 100 (10i_15_100)	Low [1]	Low [2.5]	
	9 meetings with size 15 and 90 (9m_15_90)	Low [1.6]	Low [2.6]	
	6 overlapping meetings with size 15 (6o_15)	Medium [2.4]	High [4]	
RANDOM	8 randomly generated meetings (8r)	Low [1.7]	Medium [3.5]	
	12 randomly generated meetings (12r)	Low [1.9]	Low [2.8]	

Figure 3.12: Meeting Benchmarks. Each node represents a time slot, while the edges represent meeting name and [size].

$$\text{Avg. Sched. Meetings} \triangleq \frac{\sum_{t=1}^{T} \#\text{ of meetings at } t}{\mathbb{T}}$$
$$\text{Avg. Sched. Flex} \triangleq \frac{\sum_{i=1}^{m} \#\text{ of rooms that can hold } M_i}{m} \tag{3.27}$$

The energy values are computed from 7am to 7pm, and includes the room energy of all the rooms of the building presented in Figure 3.2. The set temperature for an unoccupied room is 15.6°C and 21°C for an occupied room, with an offset of 15 minutes.

Our meeting assignment algorithms are implemented in C. To obtain the corresponding energy from the EnergyPlus simulator, a function `EnergyPlus(`M`, R)` invokes a Perl script that calculates the building energy for an input meeting schedule. The Perl script takes the building configuration (`.idf`) file as an

input, maps the rooms from the schedule to the thermal zones of the building, creates the thermostat and occupancy schedules, and invokes the EnergyPlus simulator through a batch script. A parser processes the EnergyPlus result files and calculates the cumulative HVAC energy. Measurements begin an hour before the first scheduled meeting and end an hour after the last one. This is done to ensure that the rooms are preconditioned to a minimum comfortable temperature and to reduce the effect of external factors on the HVAC energy measurements.

To solve the 0-1 ILP formulation of the meeting-room assignment problem, we use the CPLEX solver [17]. We compare its energy savings for the above benchmarks with respect to the search-based algorithms. We also examine its scalability for larger meeting room benchmarks.

3.8 Results

3.8.1 Performance of Meeting Room Assignment Algorithms

Baselines

We compare the performance of our meeting assignment algorithms, in terms of energy savings with respect to two algorithms. We use *Random Assignment* as a practical baseline. This algorithm averages the energy over 30 randomly generated assignments, each meeting the capacity and timing constraints. Furthermore, we use *Oracle* algorithm as an impractical oracle against which we compare the performance of the other algorithms. The *Oracle* algorithm per-

forms an exhaustive search of all possible n^m meeting-room combinations within the constraint boundaries, and returns the minimum energy assignment as determined by EnergyPlus. Although conflicts and mismatches may reduce the number of feasible solutions in some cases, each solution runs an EnergyPlus simulation, which makes *Oracle* computationally impractical. For instance, EnergyPlus simulations of the 1,048,576 solutions for scheduling 10 continuous non-overlapping meetings in 4 rooms requires 108 days of execution time using a 2.66GHz Intel® Xeon® 5150 processor.

We also compare our approach to the state-of-the-art *Capacity Match* algorithm, proposed by Pan et al. [126]. *Capacity Match* greedily assigns meeting M_i to a room R_j such that the difference (R_j.*capacity* - M_i.*size*) is minimized. Since conditioning a bigger room requires proportionally more HVAC energy than a smaller room, this algorithm avoids scheduling a meeting with few attendees in a larger room. However, when the room sizes are relatively similar, this algorithm often results in meeting gaps, which results in high overhead energy.

Energy Savings

The energy savings of the meeting scheduling algorithms relative to the baseline *Random Assignment* algorithm is shown in Figure 3.13. The percentage energy savings is shown in the left while the absolute energy savings is shown in the right. The energy savings are reported for five winter days. Since the active-to-sleep power ratio (β) and the time constant (τ) depend on outside weather conditions and the thermal state of the room, their actual values vary over these five days. In our model, we considered a constant β of 0.4 and τ of 5 minutes. We show in Section 6.6.4 that this assumption does not lead to large modeling

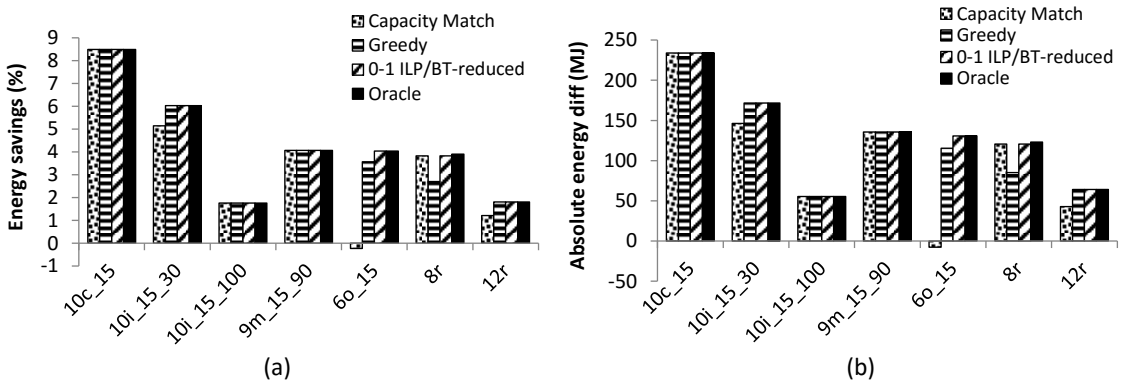


Figure 3.13: EnergyPlus energy savings with respect to the *Random Assignment* baseline for January 28th to February 1st. *BT-reduced* and *0-1 ILP* show similar energy savings.

inaccuracies.

Capacity Match assigns meetings to the room as per its capacity requirement and avoids using an oversized room. For two of the three serial synthetic benchmarks (*10c_15* and *10i_15_100*), *Capacity Match* matches the *Oracle* algorithm. However, it falls short for the *10i_15_30* serial case since it creates many gaps. Here, the difference between the room capacities is small enough that it is better to place all the meetings in the same room (R_3) to eliminate meeting gaps (and therefore the associated overhead energy). *Capacity Match* also performs poorly for *6o_15*. One of the challenges of this benchmark is that there are conflicting meetings with the same capacity starting at the same time of 8:00am. Because the capacity differences are identical, it is unable to come close to the optimal assignment, and performs worse than *Random Assignment*.

The *Greedy*, *BT-reduced*, and *0-1 ILP* algorithms perform consistently close to optimal across all the benchmarks, in part due to the energy model guiding the search for the optimum assignment. *Greedy* properly assigns all the meetings in the same room for *10i_15_30*, while it uses two rooms for *10i_15_100* creating gaps. It also intelligently assigns *6o_15* where multiple conflicting meetings

Table 3.2: Weather simulated with different β ratios.

	Outdoor Temp. Range (°C)	**Range of β during gaps**	**Weather**
Jan 30th	-31 to -21	0.18 to 0.34	Extreme
Feb 1st	-13 to -8	0 to 0.19	Less extreme
Apr 21st	0 to 1.1	0	Moderate

start at the same time, demanding the same room as their lowest cost option. However, since *Greedy* is non-backtracking, it is outperformed by *BT-reduced* for *6o_15*, *8r* and *12r*. *BT-reduced*, and *0-1 ILP* perform identical to *Oracle* for all the benchmarks. The energy savings of *BT-reduced* and *0-1 ILP* are identical over all the 7 benchmarks, and match with that of *Oracle* for six out of seven benchmarks. For *8r*, the energy savings falls short from *Oracle* by roughly 0.08%. This is primarily because *BT-reduced* and *0-1 ILP* rely on the energy model using a constant β and τ value.

3.8.2 Validating the Model Intuition

In this section, we analyze the energy savings in more detail with the aim of validating the insights gained in Section 3.6.3. We show how the model predicts the trends seen from simulating actual meeting room schedules. We also identify the meeting cases where a sophisticated *0-1 ILP* algorithm significantly improves upon simple *Random Assignment*, and when it is not worthwhile.

We first examine the energy savings over different weather situations by varying the β values. To simulate different β values, we consider the energy savings for three different days in Minneapolis, as shown in Table 3.2. Figure 3.14 shows the energy savings of *0-1 ILP* compared to the random assignment for different benchmarks and days of the year. This figure validates a number of

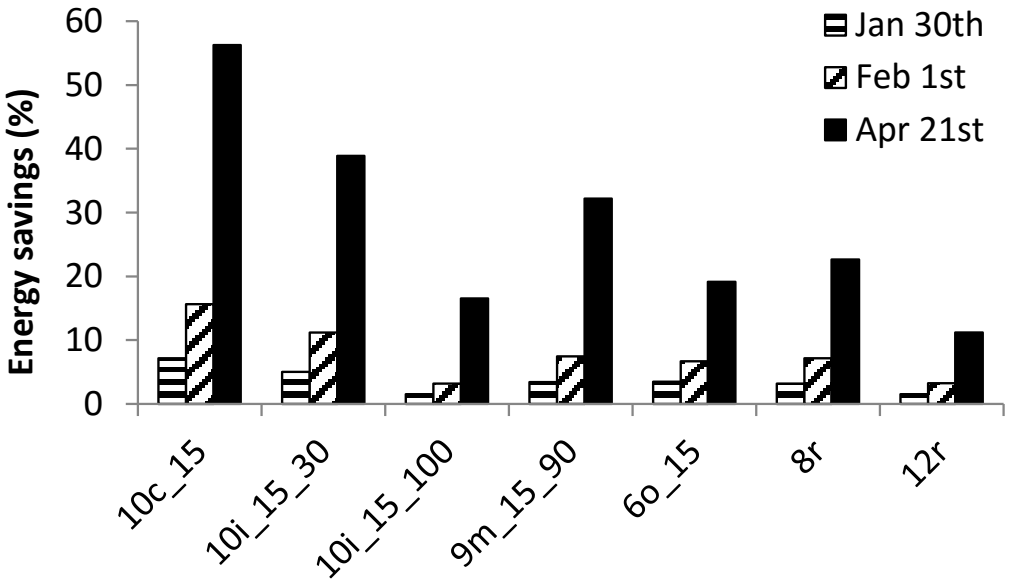


Figure 3.14: EnergyPlus energy savings with respect to the *Random Assignment* baseline for different days and benchmarks.

the insights gained from our model.

First, the effect of β is apparent by comparing the results from the different days of the year. *Apr 21st* with its very small β shows much higher energy savings than *Feb 1st* and *Jan 30th*. These latter two days incur more extreme winter, and thus spend relatively more *Sleep* energy during the meeting gaps, which significantly constrains the relative benefit of the smarter *0-1 ILP* algorithm. This matches the basic intuition derived from Figure 3.8.

Figure 3.14 also shows the effect of the number of scheduled meetings and the scheduling flexibility predicted by the model. Benchmark *10c_15* achieves the highest energy savings since it has both few scheduled meetings and high flexibility. On the other extreme, benchmark *12r* has the second highest number of scheduled meetings and a relatively low scheduling flexibility; thus, its relative energy savings is very small, especially for higher values of β, making the complexity of *0-1 ILP* difficult to justify. Benchmark *6o_15* also achieves very lit-

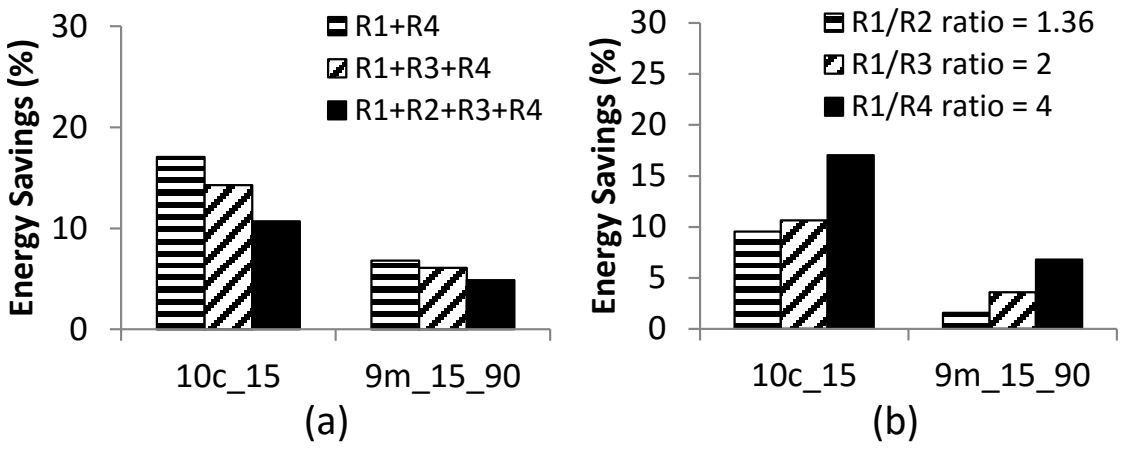


Figure 3.15: (a) Effect of additional unused rooms on the energy savings. R_2 and R_3 are unused, while the meetings are assigned only to R_1 and R_4. (b) Impact of room capacity ratios on energy savings with two available rooms.

tle savings due to its relatively high number of scheduled meetings. Thus, these results demonstrate how the basic intuition provided by the model (from Figures 3.9 and 3.10) can pinpoint the limitations of complex meeting scheduling algorithms, and the situations where they would prove most useful.

The insights provided by the model are further demonstrated in Figure 3.15, which shows the variation in energy savings as the number of empty rooms increases, as well as the capacity ratios. We consider assigning meetings only to R_1 and R_4 for two of our benchmarks, one with a single meeting assignment and a second where up to two meetings may be simultaneously assigned. For Figure 3.15(a), we show the effect of adding unused rooms R_2 and R_3 on the overall energy savings. As predicted by Figure 3.10, as the number of unused meeting rooms (n in the figure) increases, the energy savings decreases due to the added *Sleep* energy.

For Figure 3.15(b), we assume two rooms and analyze the effect on the room capacity ratio (r in Figure 3.10). As predicted by Figure 3.10 and shown in Figure 3.15(b), increasing the ratio r increases the energy savings as there is more

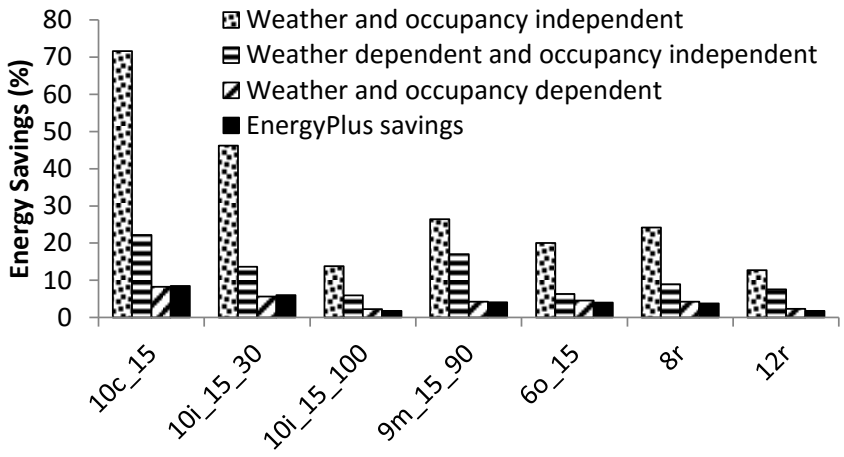


Figure 3.16: Comparison of the energy savings estimated by different models compared to EnergyPlus.

room for *Active* energy savings.

3.8.3 Comparing the Energy Savings with Prior Models

In this section, we compare the energy savings estimated by our model to that of EnergyPlus, and make a rough comparison with prior models. We simulate the meeting benchmarks from January 28^{th} to February 1^{st}, and report the energy savings from 7am to 7pm using *0-1 ILP* over the *Random Assignment* algorithm. Our *Weather and occupancy dependent* model uses values of $\beta = 0.4$, $\tau = 5$ minutes, and a γ that is linearly proportional to occupancy. The *Weather and occupancy independent* model is based on Kwak et al. [84], which assumes no energy during meeting gaps and is occupancy independent. We implement this model by setting $\beta = 0$, and $\gamma = 1$. Chai et al. consider extra energy consumption during meeting gaps, which can be modeled as sleep energy, but is occupancy independent [32]. This *Weather dependent and occupancy independent* model is constructed by setting $\beta = 0.4$, $\tau = 5$, and $\gamma = 1$.

Table 3.3: Runtime of meeting assignment algorithms. Execution Time units for Oracle are in hours:minutes:seconds unless otherwise stated.

Benchmarks	Greedy	0-1 ILP	BT-reduced		Exhaustive Search	Oracle	Total # of solns.
	Exec. Time (ms)	Exec. Time (ms)	Exec. Time (ms)	# of solns. tested	Exec. Time (ms)	Exec. Time (hh:mm:ss)	
10c_15	4.89	40	11.28	1	1630.53	108 days[3]	1,048,576
10i_15_30	4.95	90	35.69	5	390.64	24 days[3]	248,832
10i_15_100	4.72	10	5.67	1	6.25	2:23:28	1,024
9m_15_90	4.60	10	6.81	1	7.46	0:45:57	324
6o_15	4.78	20	5.01	10	5.04	0:20:26	144
8r	4.63	60	11.06	2	6.96	3:25:45	1,440
12r	4.95	30	8.23	8	7.28	3:06:06	1,296

As shown in Figure 3.16, the *Weather and occupancy independent model* incurs a large error with respect to EnergyPlus, while adding weather dependency reduces the error. Our model, which takes into account both factors, closely matches EnergyPlus, with a maximum difference of 0.5% in energy savings. This result demonstrates how the additional parameters embedded in our model greatly improve modeling accuracy.

3.8.4 Algorithm Runtime

Table 3.3 presents the CPU execution time of the algorithms running on an 2.66GHz Intel® Xeon® 5150 processor with 16GB of DRAM. We also report the runtime of an exhaustive search algorithm (*Exhaustive Search*) that searches the entire search tree without pruning. *Greedy*, *BT-reduced*, *Exhaustive Search*, and *Oracle* are written in C and compiled using gcc with -O3 optimization. We use the CPLEX solver to perform 0-1 ILP optimization.

Table 3.4: Runtime for larger meeting-room benchmarks (* indicates execution too long to complete within a reasonable timeframe).

Benchmarks		**Greedy**	**0-1 ILP**	**BT-reduced**		**Exhaustive Search**	
Rooms (*n*)	**Meetings (*m*)**	**Exec. Time**	**Exec. Time**	**Exec. Time**	**# of solns. tested**	**Exec. Time**	**# of solns. tested**
4	8	4.64 ms	0.02 s	4.78 ms	1	4.89 ms	8
6	12	4.94 ms	0.06 s	0.4 s	18	163.08 ms	51,840
8	16	Failed	0.33 s	11.43 s	13	9.7 s	2,534,400
9	18	5.06 ms	0.65 s	> 8 hrs.	-	>12 hrs.	-
10	20	5.07 ms	0.59 s	*	*	*	*
12	24	5.52 ms	6.43 s	*	*	*	*
14	28	6.34 ms	9.78 s	*	*	*	*
15	30	6.18 ms	6.05 mins	*	*	*	*

From Table 3.3, all the algorithms outperform *Oracle* simulations. This is primarily because the execution time of *Oracle* is dominated by the complex energy calculations of the EnergyPlus simulations, which the other algorithms avoid by minimizing our simplified energy model. In general, *Exhaustive Search* looks for all possible meeting-room assignments, and thus requires more time to run than *BT-reduced*. However, *BT-reduced* shows a slightly longer execution time for *8r* and *12r* because it evaluates the energy model at every partial assignment, while *Exhaustive Search* does so only for the complete assignment. Despite examining few solutions, the two benchmarks do not benefit from pruning since the further search is eliminated near the leaves of the tree. *Greedy* shows the fastest execution time due to its polynomial time complexity. *0-1 ILP* shows slightly higher runtime because of the initial pre-processing overhead of CPLEX, which is in the order of tens of milliseconds.

Table 3.4 shows the CPU execution time for randomly generated larger

[4]By inspection of the benchmark, we were able to find the optimal without simulating this many cases.

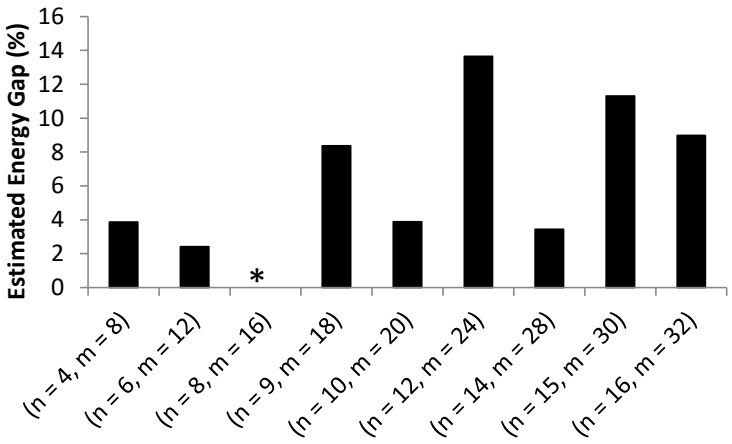


Figure 3.17: Estimated energy gap between *Greedy* and *0-1 ILP*. * indicates *Greedy* could not find an assignment.

meeting-room benchmarks to demonstrate the scalability of the proposed algorithms. All the algorithms minimize the energy model and do not run EnergyPlus simulations. The search space of these benchmarks grows non-polynomially, with n^m possible assignments (including the infeasible assignments). As expected, *Exhaustive Search* requires a prohibitively long time to complete for even moderate problem sizes. *BT-reduced* searches far fewer solutions than *Exhaustive Search* but evaluates energy for all intermediate assignments. As a result, it is faster for the first two benchmarks but slower for the benchmark of 8 rooms and 16 meetings. *0-1 ILP* benefits from the highly optimized CPLEX solver, and therefore shows better scalability than *BT-reduced* and *Exhaustive Search*. It solves benchmarks with up to 15 rooms and 30 meetings within 6 minutes. The *Greedy* algorithm scales polynomially with n and m. However, it fails to generate a solution for two benchmarks, and does not achieve global optimality.

3.8.5 Summary

To summarize, this chapter yields the following insights:

- Using the proposed energy model as a proxy for optimizing meeting room scheduling energy results in near-oracle energy savings for different meeting scenarios.
- The energy savings potential is high when the outside temperature is less extreme, and when the building has rooms with large size ratios and fewer unusable rooms. For these cases, the use of complex assignment algorithms is worthwhile.
- The energy savings potential is high for meeting scenarios with fewer simultaneously scheduled meetings and high scheduling flexibility. For these cases, the use of complex assignment algorithms is worthwhile.
- Greedy methods have the advantage of operating in polynomial time and therefore achieve the highest scalability for larger problems. However, they experience a loss in optimality and may fail to find a solution.
- The ILP formulation benefits from faster solvers and therefore achieves greater scalability for larger benchmarks compared to the backtracking-based search algorithms.

3.9 Conclusions

The energy consumption of commercial buildings is of growing worldwide concern, which calls for new approaches that analyze scheduled building occupant activities and proactively take steps to curb building energy use.

In this chapter, we address the problem of automatically assigning meetings to rooms such that the overall energy consumption is minimum. We develop an abstract energy model based on a building power characterization study, and capture the key parameters that impact meeting room scheduling energy savings. We use this model as a proxy for representing energy consumption when meetings are assigned to rooms, and propose a variety of meeting assignment algorithms. Specifically, we propose search algorithms and an ILP-based formulation to determine the energy-optimal meeting assignment.

We also characterize how various meeting, building, and weather parameters impact the potential energy savings. We further demonstrate how the model predicts when the conditions exist for significant energy savings using smart meeting room scheduling, and when it is not worthwhile. We demonstrate through EnergyPlus simulations the intuition derived from our model, and how the addition of key modeling parameters significantly improves accuracy.

Lastly, we examine the runtime performance of all the algorithms. Greedy methods have the highest scalability but experience some loss in optimality and do not guarantee a solution. Backtracking search methods are generally unscalable to large problems, despite giving globally optimum solutions. The ILP formulation benefits from fast solvers, and thus shows greater scalability than backtracking search.

CHAPTER 4

SOLAR AWARE ENERGY-EFFICIENT MEETING ASSIGNMENT

4.1 Introduction

In Chapter 3, we propose an energy model to determine the important energy factors impacting the energy savings of meeting room scheduling. Through this model, we were able to identify the situations when a complex meeting room assignment algorithm is worthwhile, and were able to accurately predict the energy savings in comparison to state-of-the-art occupancy and weather aware energy models that accounted for only outside temperature.

However, the model does not include the impact of the windows, the wall material, and most importantly – the orientation of the rooms toward the sun. From our own simulation, scheduling meetings to a bigger south-facing room of a building situated in the northern hemisphere may result in 15% energy savings over our current model, which chooses the smaller north-facing room. However, such behavior is observed only on certain days. Therefore, it is important to include the solar factors in our energy model to make the most energy efficient meeting assignments.

These solar factors, in general, exhibit a highly non-linear relationship with the outside weather conditions and with the building model. As a result, the intuitions derived from characterizing one building over a few set of days may not be applicable to a variety of meeting situations and building models. Therefore, a general methodology of accurately modeling these factors can be applicable to other building designs and weather conditions, and hence can further improve

the energy efficiency of meeting assignment.

In this chapter, we extend our proposed energy model from Chapter 3 to include solar factors. We first evaluate north- and south facing rooms, and observe a large difference in their respective active energy on certain days. Motivated by this finding, we include a solar factor in our energy model to improve meeting room scheduling. We identify the external conditions impacting the solar factor, and propose a methodology that takes the building parameters and solar factors to determine the energy-efficient meeting schedules for a given day.

To examine the performance of our proposed solar-aware energy model, we create a wide range of meeting benchmarks, and simulate a number of buildings. Our solar-aware energy model achieves near-oracle energy savings for a variety of meeting situations and buildings.

4.2 Motivation

In this section, we profile the energy behavior of a two-room building and use our findings to demonstrate the limitations of our previously proposed energy model.

4.2.1 Building Infrastructure

In this chapter, we study a two-room building with a layout shown in Figure 4.1. The walls are polished with mat sheath, which is then followed by a layer of wall insulation and half-inch gypsum. The specifications of each layer is given

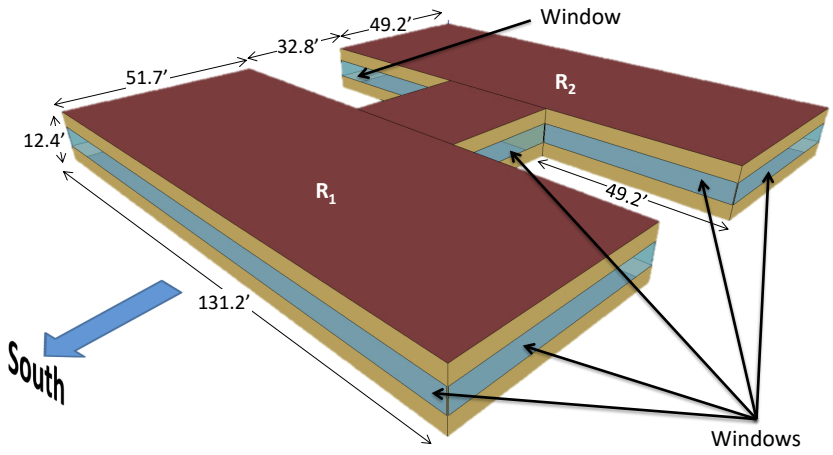


Figure 4.1: Layout of the 2-room building. Room R_1 faces south while room R_2 faces north.

in Table 4.1. The windows are made of glass, whose specifications are provided in Table 4.2. The layout is designed using the Google Sketchup Tool [11].

We use the Department of Energy's building energy simulation software EnergyPlus version 8.2 to model and characterize the energy [6]. We use the same setup that was used in Section 3.3. That is, the HVAC control system uses the default *IdealLoadsAirSystem* with negligible building infiltration. We consider a Demand Controlled Ventilation of 5 cfm outdoor airflow per person and 0.06 cfm per square feet of room to satisfy ASHRAE's 2010 ventilation standards of occupant comfort and indoor air quality [15].

The building has two rooms: room R_1 facing south and the room R_2 oriented toward the north. The room R_1 is 5% bigger than R_2. Assuming an occupant density of 12 feet square of area per person [103], the capacities of rooms R_1 and R_2 are 565 and 538, respectively. Each room maps to a unique thermal zone, and is controlled by an individual zone-level thermostat. We separate the two rooms with a corridor to minimize the impact of lateral heat transfer. Throughout this chapter, we keep the corridor at the sleep power state. The building is situated

Table 4.1: Specification of the wall material used in buildings *5p*, *25p*, and *25p_1win*.

Steel-Framed_R-13 + R-7.5 ci_Ext-wall	**Property**	**Value**
MAT-SHEATH	Roughness	Rough
	Thermal Resistance (m^2-K/W)	0.3626
	Thermal Absorptance	0.9
	Solar Absorptance	0.7
	Visible Absorptance	0.7
Wall Insulation	Roughness	MediumRough
	Thickness (m)	0.0971
	Thermal Resistance (m^2-K/W)	22.88
	Density (kg/m^3)	265
	Specific Heat (J/kg-K)	836.8
	Thermal Absorptance	0.9
	Solar Absorptance	0.7
	Visible Absorptance	0.5
1/2 GYPSUM	Roughness	Smooth
	Thickness (m)	0.0127
	Thermal Resistance (m^2-K/W)	492.125
	Density (kg/m^3)	784.9
	Specific Heat (J/kg-K)	830
	Thermal Absorptance	0.9
	Solar Absorptance	0.92
	Visible Absorptance	0.92

in Minneapolis, Minnesota. Because of its location in the northern hemisphere, R_1 receives more sunlight than R_2. We consider the winter season and model the heating energy.

4.2.2 Active Power Characterization

To understand the impact of weather conditions on the relative energy of the two rooms, we profile the active power of these rooms at full occupancy for two days with different weather characteristics.

Table 4.2: Specifications of the glass used in the windows of buildings *5p*, *25p*, *25p_1win* and *25p_mat*.

std_window_UValue_3.124_SHGC_0.4_VT_0.508	**Property**	**Value**
Theoretical Glass	Thickness (m)	0.003
	Solar Transmittance at Normal Incidence	0.3933
	Front Side Solar Reflectance at Normal Incidence	0.5567
	Back Side Solar Reflectance at Normal Incidence	0.5567
	Visible Transmittance at Normal Incidence	0.5079
	Front Side Visible Reflectance at Normal Incidence	0.4421
	Back Side Visible Reflectance at Normal Incidence	0.4421
	Infrared Transmittance at Normal Incidence	0
	Front Side Infrared Hemispherical Emissivity	0.9
	Back Side Infrared Hemispherical Emissivity	0.9
	Conductivity (W/m-K)	0.0185
	Dirt Correction Factor for Solar and Visible Transmittance	1
	Solar Diffusing	No

Figures 4.2 and 4.3 show the active power per unit room area of the north- and south-facing rooms. The figures also include sky clearness and outdoor air temperature. Larger values of sky clearness indicates a clearer sky. As shown in Figure 4.2, February 1^{st} is a colder day with occasional sunshine. As a result, we observe that the south-facing room's active power coincides with that of north-facing room, and therefore the active energy is proportional to the room capacity.

However, for February 9^{th}, which has a clearer sky and is a relatively warmer day, the south-facing room consumes lower active power than the north-facing room because it receives more sunlight. We further observe that both the rooms eventually reach zero active power. This is because we consider active power with full occupancy; the combined effect of relatively warm weather plus the heat generated by a large number of occupants coupled with a minimum demand controlled air flow rate requires no warming up the mixed air, thereby

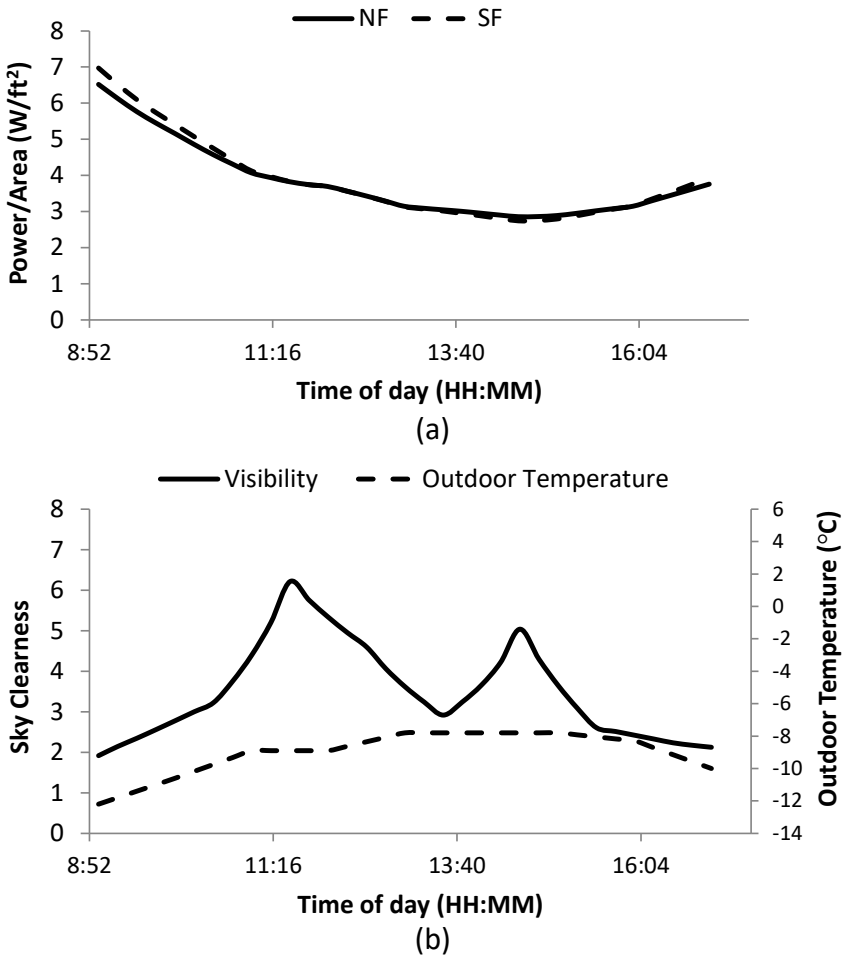


Figure 4.2: (a) Active power per unit area of different rooms, and (b) corresponding sky clearness and outdoor temperature for February 1^{st}. Larger values of sky clearness indicates a clear sky, and vice versa.

spending zero heating energy. Nevertheless, we observe that the south-facing room reaches zero active energy earlier than the north-facing room due to its orientation toward the sun.

Limitations of Solar Agnostic Energy Model

To understand how the relative meeting assignment energy savings gets affected as a consequence of weather conditions, we study three meeting mi-

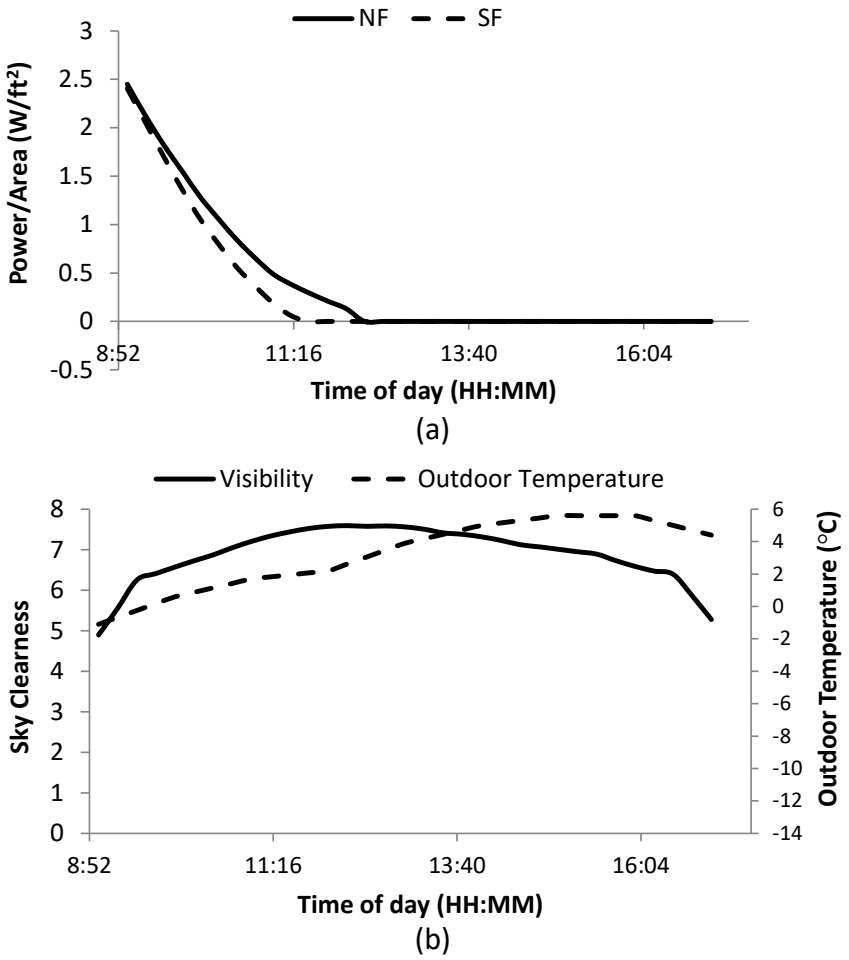


Figure 4.3: (a) Active power per unit area of different rooms, and (b) corresponding sky clearness and outdoor temperature for February 9^{th}. Larger values of sky clearness indicates a clear sky, and vice versa.

crobenchmarks presented in Table 4.3. Figure 4.4 compares the energy savings and the absolute energy difference with respect to the *Capacity Match* algorithm, presented in Chapter 3. The *Capacity Match* assignment algorithm always schedules meetings to the smaller north-facing room. As a result, we observe no energy savings on February 1^{st}. This is because the bigger south-facing room consumes proportionally more energy on February 1^{st}. However, on February 9^{th}, we get 7-15% energy savings with a maximum energy difference of 42 MJ. The EnergyPlus based assignment algorithm schedules meetings to the south-facing room, and thus shows large energy savings. For the conflicting meeting,

Table 4.3: List of microbenchmarks.

	Meeting time	Occupancy
Single	1pm-5pm	25
Serial	9am - 12noon	75
	1pm - 5pm	25
Conflict	1pm - 5pm	25
	2pm - 4pm	75

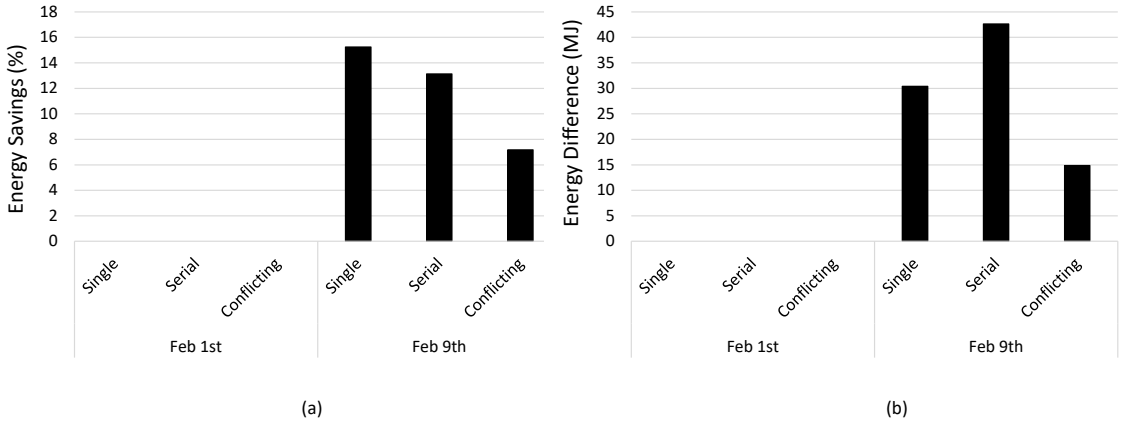


Figure 4.4: (a) Energy savings, and (b) absolute energy difference of EnergyPlus based assignment algorithm with respect to our *Solar Unaware* model presented in Chapter 3.

the *Solar Unaware* model schedules the longer meeting to the north-facing room while the EnergyPlus assignment algorithm uses the south-facing room.

Motivated by such a large gap in the energy savings between the algorithm based on our previous model and that of the EnergyPlus based approach, we include a solar factor in our energy model that captures the impact of the weather conditions. This energy model is used by our assignment algorithms to make more intelligent scheduling decisions.

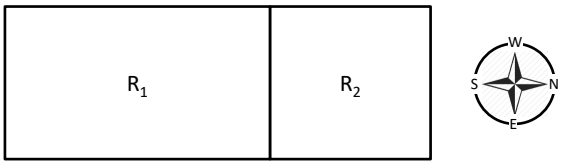


Figure 4.5: A building with room R_1 facing south, and room R_2 facing north. R_1 is r times bigger in area than R_2.

4.3 Solar-Aware Energy Model

In this section, we extend our energy model developed in Chapter 3 to include solar factors to increase the effectiveness of our meeting room assignment algorithms.

4.3.1 Solar-Aware Energy Model for a Two Room Building

Figure 4.5 shows the schematic of a building with a south-facing room R_1 and a north-facing room R_2. The room R_1 is r times bigger in size than R_2.

At timestep t, let $E_{a,sf,t}$ and $E_{sl,sf,t}$ be the active and sleep energy of the south-facing room, and $E_{a,nf,t}$ and $E_{sl,nf,t}$ be the active and sleep energy of the north-facing room. Also, there is one meeting with an occupancy of occ_t that needs to be scheduled. Recall from Chapter 3 that the active and sleep energies are a function of occupancy. Our baseline is the *Capacity Match* algorithm presented in Chapter 3, which selects the smaller north-facing room to schedule the meeting. The assignment algorithm using a solar-aware energy model selects the south-facing room to schedule the meeting. As a result, the solar-aware energy savings, $\eta_{solar,t}$, at timestep t is given by Equation 4.1.

$$\eta_{solar,t} = 1 - \frac{E_{a,sf,t}\,(occ_t) + E_{sl,nf,t}(0)}{E_{a,nf,t}\,(occ_t) + E_{sl,sf,t}(0)} \tag{4.1}$$

From Chapter 3, recall that at timestep t, β_t is the ratio of sleep to active power of a room at full occupancy, and the occupancy factor $\gamma(occ_t)$ is the ratio of active energy of a room with occupancy occ_t relative to the active energy of the same room with maximum occupancy. Let $\beta_{nf,t}$ and $\beta_{sf,t}$ correspond to the sleep to active power ratio of the north and south-facing rooms, respectively. Also, let $\gamma_{nf,t}$ and $\gamma_{sf,t}$ correspond to the occupancy factors of the north and south-facing rooms, respectively. With these definitions, we arrive at Equation 4.2.

$$\begin{aligned}\eta_{solar,t} &= 1 - \frac{\gamma_{sf,t}(occ_t)E_{a,sf,t}\left(occ_{full}\right) + \beta_{nf}E_{a,nf,t}\left(occ_{full}\right)}{\gamma_{nf,t}(occ_t)E_{a,nf,t}\left(occ_{full}\right) + \beta_{sf,t}E_{a,sf,t}\left(occ_{full}\right)} \\ &= 1 - \frac{\gamma_{sf,t}(occ_t)\dfrac{E_{a,sf,t}\left(occ_{full}\right)}{E_{a,nf,t}\left(occ_{full}\right)} + \beta_{nf,t}}{\gamma_{nf,t}(occ) + \beta_{sf,t}\dfrac{E_{a,sf,t}\left(occ_{full}\right)}{E_{a,nf,t}\left(occ_{full}\right)}}\end{aligned} \tag{4.2}$$

Without the solar effect, we know from Chapter 3 that the active energy at full occupancy is proportional to the room capacity. As a result, a solar-unaware model replaces r with $\dfrac{E_{a,sf,t}(occ_{full})}{E_{a,nf,t}(occ_{full})}$ in the above equation. However, our analysis from Section 4.2 shows that a south-facing room spends lower heating power per area than the north-facing room when the weather conditions are appropriate. Inspired by this finding, we introduce a solar factor α_t, which when multiplied by r equals $\dfrac{E_{a,sf,t}(occ_{full})}{E_{a,nf,t}(occ_{full})}$. Formally, it is defined in Equation 4.3.

$$\alpha_t \triangleq \frac{{}^{E_{a,sf,t}\left(occ_{full}\right)}\big/_{r}}{E_{a,nf,t}\left(occ_{full}\right)} \tag{4.3}$$

By combining Equations 4.3 and 4.2, we arrive at the solar-aware energy savings shown in Equation 4.4.

$$\eta_{solar,t} = 1 - \frac{\alpha_t \gamma_{sf,t}(occ_t)r + \beta_{nf,t}}{\gamma_{nf,t}(occ_t) + \alpha_t \beta_{sf,t} r} \tag{4.4}$$

For a total time of $\mathbb{T}$, the net energy savings, η_{solar} is given by:

$$\eta_{solar} = 1 - \frac{\sum_{t=1}^{\mathbb{T}} [\alpha_t \gamma_{sf,t}(occ_t)r + \beta_{nf,t}]}{\sum_{t=1}^{\mathbb{T}} [\gamma_{nf,t}(occ_t) + \alpha_t \beta_{sf,t} r]} \tag{4.5}$$

4.3.2 Generalized Solar-Aware Energy Model

Recall from Section 3.6.2 that Equation 3.25 represents the solar-unaware energy model for a building with n rooms. If α_j corresponds to the ratio of active energy per unit area of j^{th} room with that of the active energy per unit area of any particular room in reference, Equation 4.6 shows the solar-aware energy savings at timestep t. Here, variable $Y_{j,A,t}$ is a Boolean variable, which is *true* if the j^{th} room is occupied with occupancy $occ_{j,A}$ under assignment A at timestep t; and $\gamma_{j,t}$ represents the time-dependent occupancy factor for j^{th} room.

$$\eta_{solar,t} = 1 - \frac{\sum_{j=1}^{n} \left[\alpha_{j,t}\gamma_{j,t}(occ_{j,A})r_j Y_{A,j,t} + \beta_{j,t} r_j \overline{Y_{A,j,t}}\right]}{\sum_{j=1}^{n} \left[\gamma_{j,t}(occ_{j,B})r_j Y_{B,j,t} + \alpha_{j,t}\beta_{j,t} r_j \overline{Y_{B,j,t}}\right]} \tag{4.6}$$

The net energy savings, η_{solar}, over total time $\mathbb{T}$ is given by

$$\eta_{solar} = 1 - \frac{\sum_{t=1}^{\mathbb{T}} \sum_{j=1}^{n} \left[\alpha_{j,t}\gamma_{j,t}(occ_{j,A})r_j Y_{A,j,t} + \beta_{j,t} r_j \overline{Y_{A,j,t}}\right]}{\sum_{t=1}^{\mathbb{T}} \sum_{j=1}^{n} \left[\gamma_{j,t}(occ_{j,B})r_j Y_{B,j,t} + \alpha_{j,t}\beta_{j,t} r_j \overline{Y_{B,j,t}}\right]} \tag{4.7}$$

4.4 Modeling the Solar Factor

In this section, we present a methodology that takes the building parameters and solar factors to determine energy-efficient meeting schedules for a given

day. To model the solar factor with external weather conditions, we describe the building specifications with an EnergyPlus input file with intended construction material and targeted location. After this step, we profile its active behavior for a representative set of days along with the weather conditions. Based on the individual room energy profile, we develop prediction models to predict the solar factor, indicating when a bigger south-facing room consumes less energy than a north-facing room, and by what magnitude. We use this solar factor α to make weather-aware meeting assignments. This chapter focuses only on modeling the solar factor. A similar method can be applied to model the active to sleep power ratio β and the occupancy factor γ. In our results presented later in Section 4.6, we capture β and γ directly from the EnergyPlus simulations, which implicitly captures the impact of weather on these factors.

4.4.1 Modeling the Real-valued Solar Factor (α)

To schedule the meetings to their appropriate rooms according to the outside solar conditions, we need to model the solar factor α. Using EnergyPlus simulations, we identify the most correlated factors, listed in Table 4.4. Among the many weather variables available from EnergyPlus, we selected the ones that show less repetition over the days and correlate highly with α. For instance, angles such as solar azimuth, solar incident, and solar hour angles do not vary over the days once a building and its location is fixed. Other weather variables, such as wind speed and direction, air pressure, rain status, humidity and sky brightness did not correlate well with α, while other factors, such as Outdoor Air Enthalpy and Infrared Radiation Rate correlated well with outside temperature.

Table 4.4: Weather factors to model the solar factor.

Weather Factor	**Definition**
Site Outdoor Air Drybulb Temperature (°C)	Temperature of outdoors measured by a thermometer freely exposed to the air but shielded from radiation and moisture.
Direct Solar Radiation Rate per Area (W/m^2)	Amount of solar radiation received within a 5.7° field of view centered on the sun.
Daylight Model Sky Clearness	Clearness of sky. Sky clearness close to one corresponds to an overcast sky. Sky clearness greater than six is a clear sky.
Beam Solar Radiation Luminous Efficacy (lum/W)	A measure of the visible light content of beam solar radiation; equal to the number of lumens per watt of beam solar radiation. Depends on atmospheric conditions (moisture, turbidity, cloudiness) and solar altitude.

Even using our most-correlated weather factors, the correlation coefficient we obtain for these selected factors is less than 0.4. This is because of the highly non-linear dependency of energy with weather. Since room energy also depends on its internal thermal state, a prediction model simply based on current weather factors that does not consider the room's thermal state may not predict α accurately. In fact, our attempt to model α using machine learning algorithms including regression trees [30], random forest [29], support vector machines [39], and neural network [68] failed to give good accuracy.

While predicting the actual magnitude of α is important to estimate the energy savings, with respect to energy efficient meeting assignment, a binary factor to indicate which room consumes more active energy for a two-room building is sufficient. Furthermore, the prediction models predicting the binary decision gives good accuracy because it is no longer predicting the actual magnitude

of α. As a result, we discretize our solar factor into an another Boolean parameter, called δ shown in Equation 4.8. Here, a *one* indicates that α is greater or equal to one, which represents a south-facing room consuming more active energy per unit area than the north-facing, and vice versa. When δ_t equals one, our model falls back to the energy model proposed in Chapter 3. When δ_t equals zero, it compares the sleep factor of north-facing room, $\beta_{nf,t}$ and the occupancy factor, $\gamma_{nf,t}(occ_t)$ to make a decision.

$$\delta_t = \begin{cases} 1 & \alpha_t \geq 1 \\ 0 & \text{otherwise} \end{cases} \tag{4.8}$$

Discretizing α to δ may lose some energy-efficiency because it cancels the active and sleep energy term of the south-facing room when δ_t becomes a zero, that is when the active energy of the south-facing room is smaller than that of the north-facing room. Although including the respective sleep energies indicate the north-facing room is more efficient, the model effectively compares β and γ to make an assignment. Since β is always smaller than γ, a δ based energy model would inefficiently assign meetings to the south-facing room, thereby losing energy savings. However, we show later in Section 4.6 that our approximation of real-valued α to a Boolean δ only slightly reduces energy efficiency and works effectively for a wide range of meeting benchmarks and buildings.

Using the above transformation, we use Equation 4.9 to assign meetings to rooms.

$$\eta_{solar,t} = 1 - \frac{\delta_t \gamma_{sf,t}(occ_t) r + \beta_{nf,t}}{\gamma_{nf,t}(occ_t) + \delta_t \beta_{sf,t} r} \tag{4.9}$$

Equation 4.10 shows the net energy savings for the time T.

$$\eta_{solar} = 1 - \frac{\sum_{t=1}^{\mathbb{T}}[\delta_t \gamma_{sf,t}(occ_t)r + \beta_{nf,t}]}{\sum_{t=1}^{\mathbb{T}}[\gamma_{nf,t}(occ_t) + \delta_t \beta_{sf,t} r]} \tag{4.10}$$

For a building with n rooms, Equation 4.11 shows the discretized solar-aware energy savings at timestep t.

$$\eta_{solar,t} = 1 - \frac{\sum_{j=1}^{n}\left[\delta_{j,t}\gamma_{j,t}(occ_{j,A})r_j Y_{A,j,t} + \beta_{j,t} r_j \overline{Y_{A,j,t}}\right]}{\sum_{j=1}^{n}\left[\gamma_{j,t}(occ_{j,B})r_j Y_{B,j,t} + \delta_{j,t}\beta_{j,t} r_j \overline{Y_{B,j,t}}\right]} \tag{4.11}$$

For the total time $\mathbb{T}$, net energy savings using discretized solar factor is shown in Equation 4.12.

$$\eta_{solar} = 1 - \frac{\sum_{t=1}^{\mathbb{T}}\sum_{j=1}^{n}\left[\delta_{j,t}\gamma_{j,t}(occ_{j,A})r_j Y_{A,j,t} + \beta_{j,t} r_j \overline{Y_{A,j,t}}\right]}{\sum_{t=1}^{\mathbb{T}}\sum_{j=1}^{n}\left[\gamma_{j,t}(occ_{j,B})r_j Y_{B,j,t} + \delta_{j,t}\beta_{j,t} r_j \overline{Y_{B,j,t}}\right]} \tag{4.12}$$

4.4.2 Predicting the Boolean Solar Factor (δ)

To predict δ, we use decision trees. The decision trees are trained using the Iterative Dichotomiser 3 (ID3) algorithm with information gain as the attribute selection criterion [133]. We avoid overfitting by pre- and post-pruning. We pre-prune such that the minimum instances in the leaves of the tree equal two. We post-prune by recursively merging the leaves with the same majority class to achieve an m-estimate of two. We used the Orange machine learning toolkit to generate our decision trees [43].

We create an individual decision tree for each type of building we simulate. Based on the nature of a building, we identify which decision tree model to use to assign meetings to rooms. The accuracy of our model is presented later in Section 4.6.

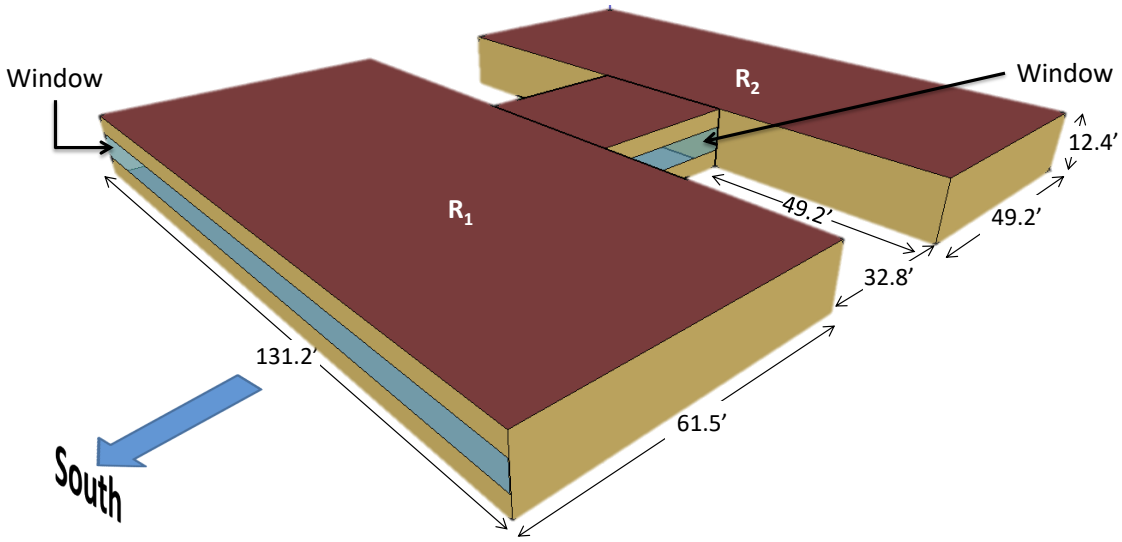


Figure 4.6: Layout of the 2-room building with a different window style and 25% larger south-facing room. Room R_1 faces south while room R_2 faces north.

4.5 Experimental Setup

To evaluate the performance of our proposed energy model and our methodology, we develop numerous benchmarks and building situations. In this section, we first describe the buildings we simulate and the meeting benchmarks we evaluate, followed by a comparison of energy models.

4.5.1 Building Models

The building layout that we use in this chapter is shown in Figure 4.1 of Section 4.2. We consider the location Minneapolis and we simulate the peak winter period. Our heating set-point is 21°C when meetings are assigned to a room and 15.6°C otherwise. We only consider the heating energy of rooms R_1 and R_2, and keep the in-between corridor unused. To examine the performance of our methodology for various buildings, we create the following configurations:

Table 4.5: Specification of wall material used in building *25p_mat*.

Standard_Int-Wall	**Property**	**Value**
1/2 GYPSUM	Roughness	Smooth
	Thickness (m)	0.0127
	Thermal Resistance (m^2-K/W)	492.125
	Density (kg/m^3)	784.9
	Specific Heat (J/kg-K)	830
	Thermal Absorptance	0.9
	Solar Absorptance	0.92
	Visible Absorptance	0.92
1/2 GYPSUM	Roughness	Smooth
	Thickness (m)	0.0127
	Thermal Resistance (m^2-K/W)	492.125
	Density (kg/m^3)	784.9
	Specific Heat (J/kg-K)	830
	Thermal Absorptance	0.9
	Solar Absorptance	0.92
	Visible Absorptance	0.92

- **5p:** In this building model, we make the south-facing room 5% bigger than the north-facing room, with window placement shown in Figure 4.1.
- **25p:** This building has the south-facing room 25% bigger than the north-facing room, with window placement shown in Figure 4.1. The dimensions of this building matches with Figure 4.6.
- **25p_1win:** This building has a 25% bigger south-facing room than its north-facing room. However, its windows are placed as shown in Figure 4.6.
- **25p_mat:** This building has a 25% bigger south-facing room than its north-facing room, with dimensions that of Figure 4.6 and window placement of Figure 4.1. Here, the wall material of Table 4.5 is used.

All of the buildings use the glass material of Table 4.2. Except for building *25p_mat*, the specifications of the material used in all the building walls is shown

Table 4.6: List of *single* meeting micro-benchmarks. The shaded rows are the days when scheduling meetings to the south-facing room consumes less overall energy than the north-facing room.

Date	Meeting Time	Occupancy
9-Jan	1pm-5pm	25
26-Jan	1pm-5pm	25
1-Feb	1pm-5pm	25
9-Feb	1pm-5pm	25
10-Feb	1pm-5pm	25
1-Mar	1pm-5pm	25
4-Mar	1pm-5pm	25
20-Mar	1pm-5pm	25

in Table 4.1. The buildings *25p_1win* and *25p_mat* have higher thermal inertia. This is because the *25p_1win* lacks the side windows of *25p*, which do not contribute to the greenhouse effect since they do not face directly toward the sun. Instead, the window area is replaced by a high-resistance wall, requiring less heating. The building *25p_mat* has a double layer of high-resistance gypsum compared to *25p*, which provides more thermal insulation.

4.5.2 Benchmarks

We first consider a series of micro-benchmarks to evaluate the energy savings of our proposed energy model. Table 4.6 shows a list of micro-benchmarks containing the same meeting on different days. Here, the shaded rows indicate the days when scheduling meetings to the south-facing room consumes less overall energy than the north-facing room. For the unshaded rows, scheduling the north-facing room is more energy-efficient. The objective of evaluating these benchmarks is to examine whether our algorithms can identify the most energy-

Table 4.7: List of (a) *serial* and (b) *conflict* meeting microbenchmarks. The shaded rows are the days when scheduling meetings to the south-facing room consumes less overall energy than the north-facing room.

Date	Meeting Time	Occupancy
9-Jan	9am - 12noon	25
	1pm - 5pm	75
26-Jan	9am - 12noon	75
	1pm - 5pm	25
10-Feb	9am - 12noon	75
	1pm - 5pm	25
4-Mar	9am - 12noon	25
	1pm - 5pm	75

(a) *Serial* meeting microbenchmarks.

Date	Meeting Time	Occupancy
9-Jan	1pm - 5pm	75
	2pm - 4pm	25
26-Jan	1pm - 5pm	25
	2pm - 4pm	75
10-Feb	1pm - 5pm	25
	2pm - 4pm	75
4-Mar	1pm - 5pm	75
	2pm - 4pm	25

(b) *Conflict* meeting microbenchmarks.

efficient meeting placement from external weather factors. We also examine a series of *serial* micro-benchmarks, shown in Table 4.7(a), which have two non-conflicting meetings each day but with different occupancies. Moreover, we also evaluate a set of *conflict* micro-benchmarks, presented in Table 4.7(b) containing conflicting meetings with different meeting lengths and occupancies. In this chapter, we consider occupancies of 25 and 75, and assume that both the rooms in our building can fit these meetings.

Finally, inspired from our microbenchmarks, we randomly create ten meeting benchmarks shown in Tables 4.8 and 4.9, to evaluate the robustness of our algorithms for different buildings and energy models. In these tables, the shaded dates are the days when scheduling meetings to the south-facing room consumes less overall energy than the north-facing room. The shaded cells indicate that meetings are assigned to south-facing room, while the unshaded cells represent meeting assigned to the north-facing room by the *EnergyPlus* assignment.

Table 4.8: List of meeting benchmarks created randomly from the micro-benchmarks. The shaded cells of the first column indicate that scheduling meetings to the south-facing room consumes less overall energy than the north-facing room. For all other columns, each cell contains a list of meetings, with their start and end times and its number of occupants. For the benchmark columns, the shaded cells indicate that meetings are assigned to the south-facing room, while the unshaded cells represent meeting assignment to the north-facing room by the *EnergyPlus* assignment (continued to Table 4.9).

Date	b0	b1	b2	b3	b4
9-Jan	1pm - 5pm, 75	1pm - 5pm, 75	1pm - 5pm, 75	9am - 12noon, 75	9am - 12noon, 75
	2pm - 4pm, 25			1pm - 5pm, 25	1pm - 5pm, 75
26-Jan	9am - 12noon, 25	1pm - 5pm, 75	9am - 12noon, 75	1pm - 5pm, 75	1pm - 5pm, 75
	1pm - 5pm, 75	2pm - 4pm, 25	1pm - 5pm, 25	2pm - 4pm, 25	
1-Feb	9am - 12noon, 75	9am - 12noon, 25	9am - 12noon, 75		9am - 12noon, 25
	1pm - 5pm, 75	1pm - 5pm, 25	1pm - 5pm, 25		1pm - 5pm, 75
9-Feb	1pm - 5pm, 25	9am - 10am, 75	9am - 12noon, 25	1pm - 5pm, 25	1pm - 5pm, 75
	2pm - 4pm, 75	1pm - 5pm, 25	1pm - 5pm, 75	2pm - 4pm, 75	2pm - 4pm, 25
10-Feb	1pm - 5pm, 75	1pm - 5pm, 75	9am - 12noon, 75	9am - 12noon, 25	
		2pm - 4pm, 25	1pm - 5pm, 25	1pm - 5pm, 25	
1-Mar	1pm - 5pm, 25	9am - 12noon, 75	9am - 12noon, 75	9am - 12noon, 75	1pm - 5pm, 25
	2pm - 4pm, 75	1pm - 5pm, 75	1pm - 5pm, 75	1pm - 5pm, 25	2pm - 4pm, 75
4-Mar	9am - 12noon, 25	9am - 12noon, 25	9am - 12noon, 75	1pm - 5pm, 75	9am - 12noon, 25
	1pm - 5pm, 75	1pm - 5pm, 25	1pm - 5pm, 25	2pm - 4pm, 25	1pm - 5pm, 75
20-Mar	1pm - 5pm, 75	1pm - 5pm, 75	9am - 10am, 75	9am - 12noon, 25	9am - 10am, 75
			1pm - 5pm, 75	1pm - 5pm, 75	1pm - 5pm, 75

4.5.3 Algorithms

In this chapter, we consider the *Capacity-Match* algorithm, presented in the Section 3.8, as the baseline. This algorithm schedules meeting to the smallest room in which the meeting will fit. We compare *Capacity-Match* with our meeting assignment algorithm using different energy models. In all of our proposed algorithms, we use the sleep-to-active power ratio (β) and the occupancy factor (γ) directly from the EnergyPlus simulations. The model *Solar_Unaware* uses the same energy model proposed in Chapter 3 with a solar factor $\alpha = 1$.

Table 4.9: (Continued from Table 4.8) List of meeting benchmarks created randomly from the micro-benchmarks. The shaded cells of the first column indicate that scheduling meetings to the south-facing room consumes less overall energy than the north-facing room. For all other columns, each cell contains a list of meetings, with their start and end times and its number of occupants. For the benchmark columns, shaded cells indicate that meetings are assigned to the south-facing room, while the unshaded cells represent meeting assignment to the north-facing room by the *EnergyPlus* assignment.

Date	b5	b6	b7	b8	b9
9-Jan	1pm - 5pm, 75		9am - 12noon, 25 1pm - 5pm, 75	9am - 12noon, 25 1pm - 5pm, 25	9am - 12noon, 75 1pm - 5pm, 75
26-Jan		1pm - 5pm, 75		9am - 12noon, 25 1pm - 5pm, 25	9am - 12noon, 25 1pm - 5pm, 25
1-Feb	1pm - 5pm, 75 2pm - 4pm, 25	9am - 12noon, 75 1pm - 5pm, 25	1pm - 5pm, 75 2pm - 4pm, 25	1pm - 5pm, 75 2pm - 4pm, 25	9am - 12noon, 25 1pm - 5pm, 25
9-Feb	9am - 10am, 25 1pm - 5pm, 75	9am - 12noon, 25 1pm - 5pm, 75	9am - 10am, 75 1pm - 5pm, 75	9am - 10am, 25 1pm - 5pm, 25	1pm - 5pm, 75 2pm - 4pm, 25
10-Feb	1pm - 5pm, 75	1pm - 5pm, 75 2pm - 4pm, 25	1pm - 5pm, 75 2pm - 4pm, 25	1pm - 5pm, 25 2pm - 4pm, 75	
1-Mar	9am - 12noon, 25 1pm - 5pm, 75	1pm - 5pm, 75	1pm - 5pm, 75 2pm - 4pm, 25	1pm - 5pm, 75	9am - 12noon, 25 1pm - 5pm, 75
4-Mar	1pm - 5pm, 25	1pm - 5pm, 75 2pm - 4pm, 25	9am - 12noon, 75 1pm - 5pm, 25	1pm - 5pm, 25	9am - 12noon, 75 1pm - 5pm, 25
20-Mar	9am - 10am, 25 1pm - 5pm, 25	9am - 12noon, 75 1pm - 5pm, 25	9am - 10am, 25 1pm - 5pm, 25	1pm - 5pm, 25	9am - 10am, 25 1pm - 5pm, 75

The algorithm *Fully_Aware* uses a δ that is predicted using a decision tree. For this algorithm, the decision tree is trained using the weather data of January 1^{st} to April 30^{th} and the heating energy is obtained from EnergyPlus simulations. The training set includes the energy data of the days used in our meeting benchmarks. The algorithm *25%_Unaware* uses a decision tree trained on 75% of the heating energy data from January 1^{st} to April 30^{th}, inclusive of some of the hourly energy data of the days used in our meeting benchmarks. The algorithm *Date_Unaware* uses the decision tree that is trained on all the energy data from January 1^{st} to April 30^{th}, except for the days used in our meeting bench-

marks. This approach examines the performance of our model to days not used in training, but whose weather forecasts are available.

Finally, the *Energy_Model* method uses our solar-aware energy model using real-valued α, while the *EnergyPlus* method uses EnergyPlus simulation results to determine the energy-optimum meeting assignment. The *EnergyPlus* method is an oracle that shows the maximum energy savings potential of meeting assignment.

4.6 Results

In this section, we demonstrate the energy savings achieved by our proposed solar-aware energy model. All of our algorithms are aware of outside weather conditions, except *Solar_Unaware*, which considers only the outdoor air temperature. We also use a decision tree to predict the discretized solar factor δ. We first show its prediction accuracy and then compare its energy savings with the *Energy_Model* approach using α and with that of an oracle using the EnergyPlus simulations.

4.6.1 Prediction Accuracy

Table 4.10 shows the accuracy of our decision tree model in predicting the discretized solar factor δ. Our *Fully_Aware* scheme trains on the entire energy data, and therefore, we report its accuracy in terms of 10-fold cross validation. The *25%_Unaware* model learns on 75% of the randomly picked data. We report its training and test accuracy. Finally, the *Date_Unaware* scheme trains on the en-

Table 4.10: Percentage accuracy of our decision tree predicting δ.

Room	Fully Aware (10-fold C.V.)	25% Unaware		Date Unaware	
		Train	Test	Train	Test
5p	97.1	99.3	97.5	99.2	91.5
25p	97.1	99.3	97.5	99.2	91.5
25p_1win	97.2	99.4	97.4	99.3	91.2
25p_mat	97.2	99.1	96.5	99.5	89.6

tire energy data except for the days of our meeting benchmarks. As shown in Table 4.10, we achieve around 90% accuracy for the unknown days.

4.6.2 Energy Savings

In this section, we first show the energy savings and absolute energy difference for a series of meeting microbenchmarks. We then later show the energy savings for meeting benchmarks comprising multiple microbenchmarks.

Meeting Microbenchmarks

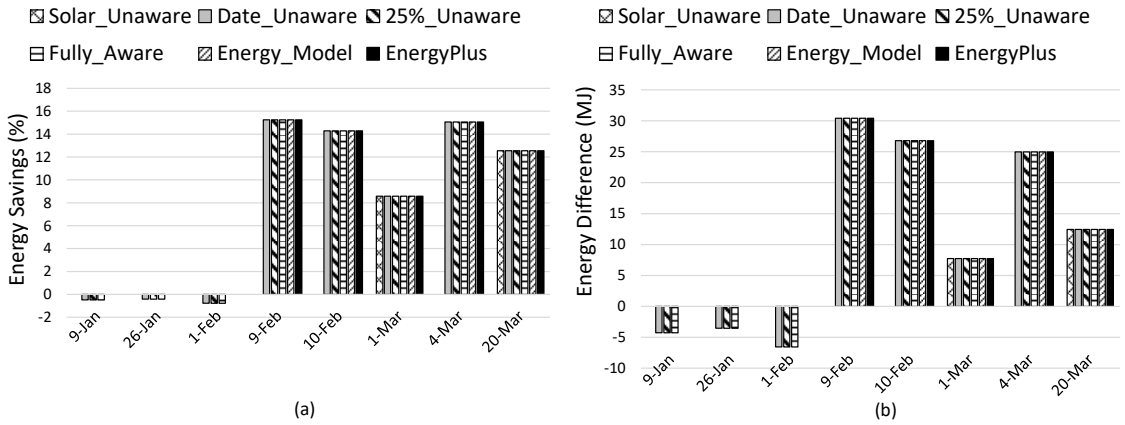


Figure 4.7: (a) Percentage and (b) absolute energy difference for *single* meeting microbenchmarks with respect to the *Capacity Match* baseline for building *5p*.

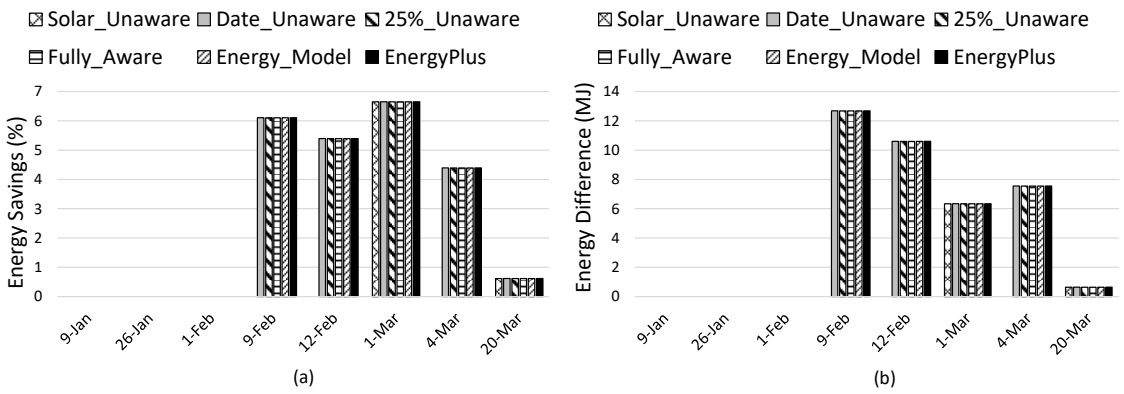


Figure 4.8: (a) Percentage and (b) absolute energy difference for *single* meeting microbenchmarks with respect to the *Capacity Match* baseline for building *25p*.

Single Meeting Figures 4.7 and 4.8 show the energy savings and absolute energy difference of our *single* meeting microbenchmarks for the buildings *5p* and *25p*, respectively. The savings are with respect to the *Capacity Match* baseline. From the above figure, the *Energy_Model* approach that uses the real-valued α achieves *EnergyPlus* energy savings for the days when the south-facing room is more energy-efficient than the north-facing one. This indicates that our energy model successfully captures the energy savings on days when the south-facing room consumes lower energy than the north-facing room.

Comparing the buildings *5p* and *25p*, the energy savings and absolute energy difference is larger for *5p* than *25p*. This is because the building *5p* has a relatively smaller room size ratio. The larger south-facing room in *25p* consumes more energy because of its size, and thus the impact of scheduling meetings to a similarly sized (as the north-facing room) south-facing room provides more energy savings than a larger south-facing room.

However, for the building *5p*, we observe that the models *Date_Unaware*, *25%_Unware* and *Fully_Aware* using Boolean-valued δ schedule meetings to the south-facing room on January 9^{th} and 26^{th} despite the fact that using the north-

facing room is overall more energy-efficient. This result, however, is not observed for the building *25p*. This is because the building *5p*'s south-facing room consumes less active energy than its north-facing room for these two days, resulting in a δ value of zero. This cancels the active and sleep energy term of the south-facing room in Equation 4.9, making it appear that scheduling to the south-facing is more energy efficient, as explained in Section 4.4. The *Energy_Model* approach uses the real-valued α to preserve the active and sleep energy term of the south-facing room in Equation 4.9 and therefore, uses the north-facing room. For the building *25p*, the bigger south-facing room consumes more energy than the north-facing room because the size ratio is significant. Because of this, all the solar-aware models schedule the meeting to the north-facing room.

For March 1^{st} and March 20^{th}, the *Solar_Unaware* model intelligently schedules the meeting to the south-facing room despite being unaware of the solar effect for both *5p* and *25p*. For these two days, the outside temperature is relatively warm. As a result, the south-facing room consumes zero active power. Consequently, its occupancy factor γ becomes zero, whereas the γ for the north-facing room remains substantial. As a result, the *Solar_Unaware* model chooses the south-facing room because it anticipates zero active energy for the south-facing room. This is also seen in buildings *25p_1win* and *25p_mat*, shown in Figures 4.9 and 4.10.

For buildings *25p_1win* and *25p_mat*, shown in Figures 4.9 and 4.10, the results of January 9^{th}, January 26^{th} and February 1^{st} are similar to that of *5p*. This is because both of these buildings show higher active energy for its north-facing room. The reason for this behavior is because both of these buildings con-

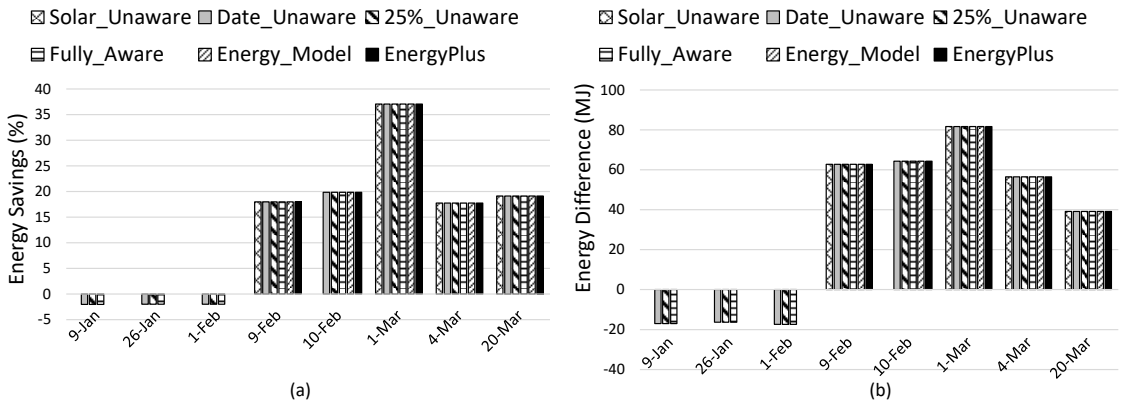


Figure 4.9: (a) Percentage and (b) absolute energy difference for *single* meeting microbenchmarks with respect to the *Capacity Match* baseline for building *25p_1win*.

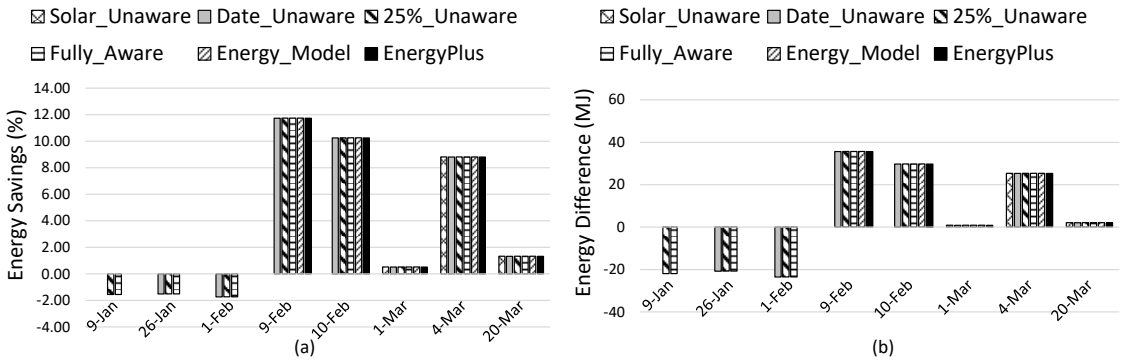


Figure 4.10: (a) Percentage and (b) absolute energy difference for *single* meeting microbenchmarks with respect to the *Capacity Match* baseline for building *25p_mat*.

serve heat more than building *25p*. Recall from Figure 4.6 that *25p_1win* has only one window with its side windows (not directly facing toward the sun) replaced by a thermally insulated wall. As a result, it requires lower heat than *25p*. The building *25p_mat* has higher wall resistance because of two layers of high-resistance gypsum, resulting in higher thermal storage. Because of these reasons, *25_1win* and *25p_mat* shows both higher energy savings and higher energy difference than *25p*.

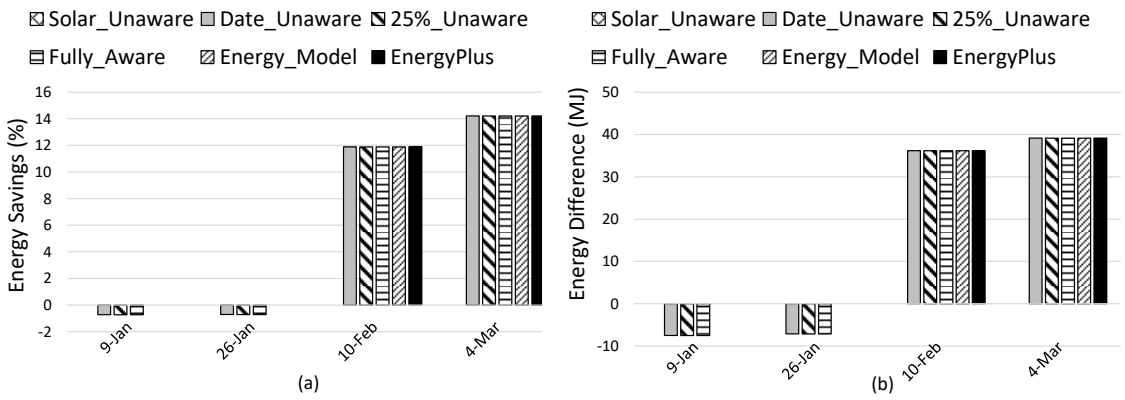


Figure 4.11: (a) Percentage and (b) absolute energy difference for *serial* meeting microbenchmarks with respect to the *Capacity Match* baseline for building *5p*.

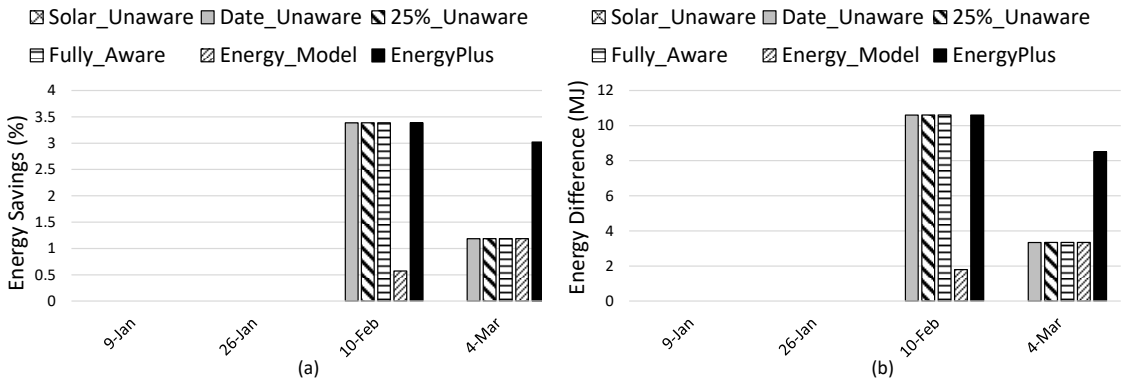


Figure 4.12: (a) Percentage and (b) absolute energy difference for *serial* meeting microbenchmarks with respect to the *Capacity Match* baseline for building *25p*.

Serial Meetings Figures 4.11 and 4.12 show the energy savings and absolute difference for buildings *5p* and *25p* for serial benchmarks. In Figure 4.11, our solar-aware models correctly assign meetings to the south-facing room for February 10^{th}, and March 4^{th}. It uses the south-facing room for the other two days because of the reasons mentioned before, and therefore consumes 1% extra energy.

For *25p*, on February 10^{th}, *Energy_Model* achieves an energy savings of 0.6%, while *EnergyPlus* achieves 3.4% savings. This is because *EnergyPlus* assigns the first 9am - noon meeting to the north-facing room, since the larger south-facing room consumes more energy during the morning. *Energy_Model*, however, uses

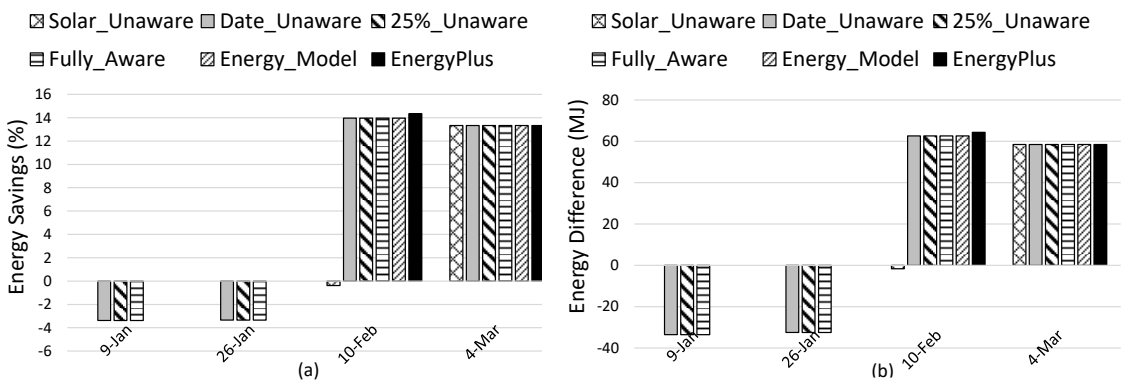


Figure 4.13: (a) Percentage and (b) absolute energy difference for *serial* meeting microbenchmarks with respect to the *Capacity Match* baseline for building *25p_1win*.

the south-facing room to schedule both the meetings. The reason is that *Energy_Model* finds the γ value between 9am - noon for the north-facing room to be more than one, which makes it disregard that room as an energy-efficient choice. The reason γ becomes greater than one is because the active energy of the north-facing room when fully occupied is very small due to warmer weather conditions along with the heat produced by the occupants. This is contrary to our findings from Chapter 3 because of the short timestep and extreme outside weather conditions. Our δ based energy models, however, match the *EnergyPlus* savings despite working with a γ greater than one. This is because δ becomes zero for the second meeting nullifying the effect of increased γ from the first meeting, thereby making an energy-optimal assignment. This situation, however, is not found in buildings *25p_1win* and *25p_mat*, as shown in Figures 4.13 and 4.14. For these two buildings, *Solar_Unaware* shows some energy savings because of the reasons mentioned previously.

Conflicting Meetings Figures 4.15, 4.16, 4.17 and 4.18 show the energy savings and energy difference for the four buildings for *conflicting* meeting mi-

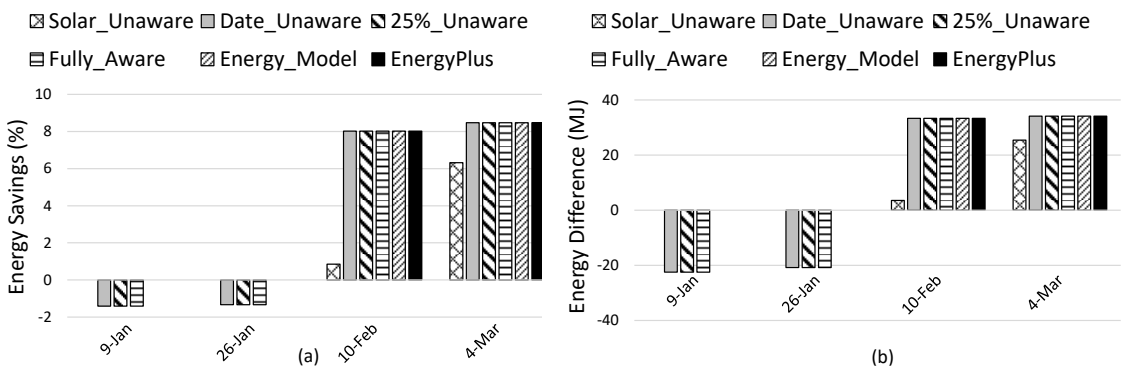


Figure 4.14: (a) Percentage and (b) absolute energy difference for *serial* meeting microbenchmarks with respect to the *Capacity Match* baseline for building *25p_mat*.

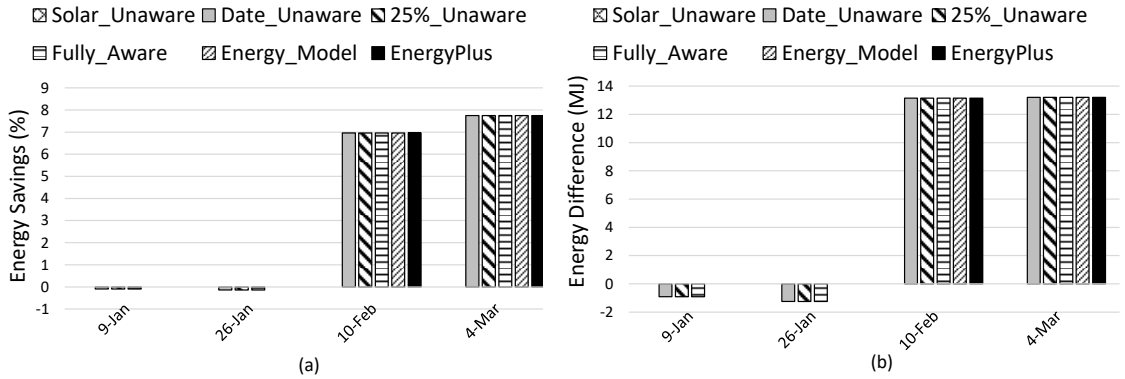


Figure 4.15: (a) Percentage and (b) absolute energy difference for *conflicting* meeting microbenchmarks with respect to the *Capacity Match* baseline for building *5p*.

crobenchmarks. For these meetings, all of our solar-aware models match the *EnergyPlus* savings, except for January 9^{th} and January 26^{th} for buildings *5p* and *25p_1win*, and January 9^{th} for *25p_mat*, because of the reasons mentioned above. For these situations, our model consumes less than 1% extra energy than *EnergyPlus*.

Surprisingly, for January 26^{th}, all of our solar-aware models match *EnergyPlus* for *25p_mat* despite predicting the south-facing room to be the most energy-efficient choice for the majority of the day. This is because the prediction model

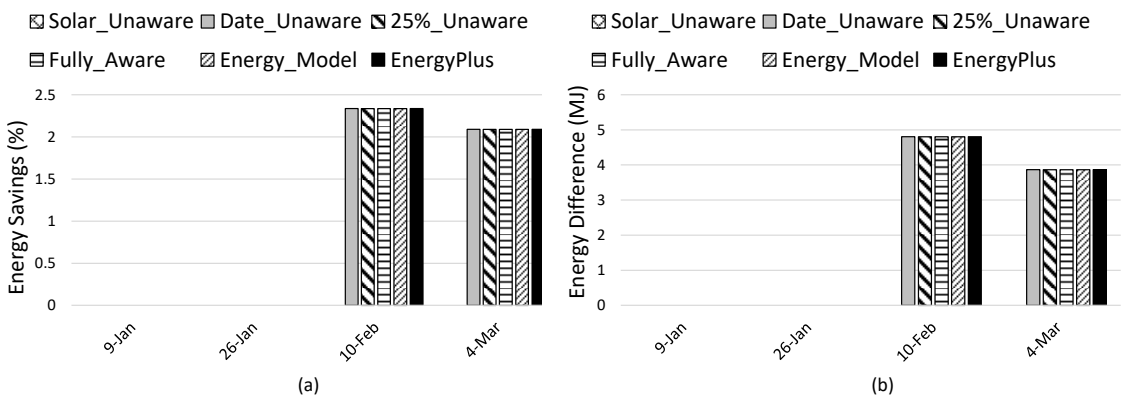


Figure 4.16: (a) Percentage and (b) absolute energy difference for *conflicting* meeting microbenchmarks with respect to the *Capacity Match* baseline for building *25p*.

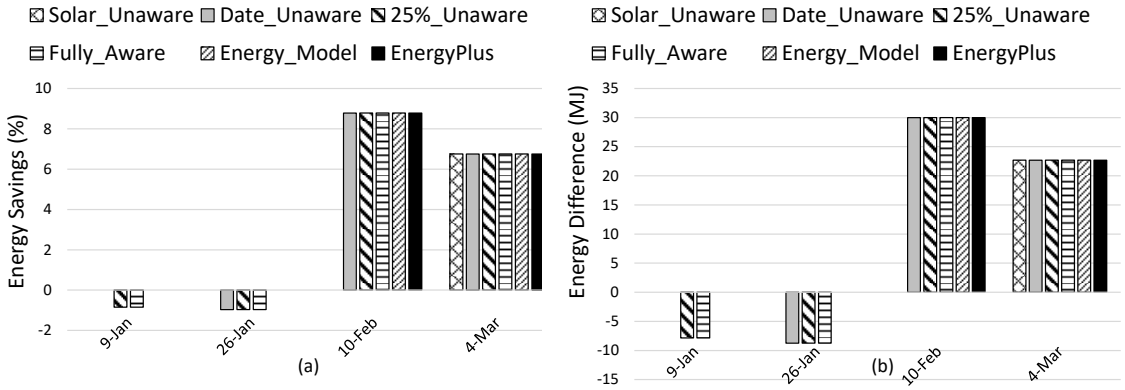


Figure 4.17: (a) Percentage and (b) absolute energy difference for *conflicting* meeting microbenchmarks with respect to the *Capacity Match* baseline for building *25p_1win*.

of *25p_mat* predicts the north-facing room to be energy-efficient for one extra timestep, which results in using the north-facing room to schedule the meeting with more occupants. This effect is prominent only in *25p_mat*. Lastly, the *Solar_Unaware* approach matches *EnergyPlus* for March 4^{th} for buildings *25p_1win* and *25p_mat* because of the aforementioned reasons.

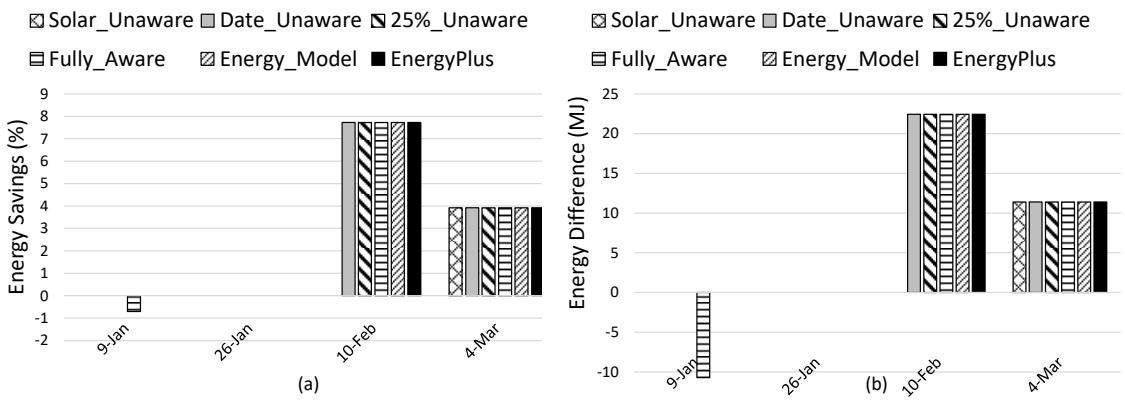


Figure 4.18: (a) Percentage and (b) absolute energy difference for *conflicting* meeting microbenchmarks with respect to the *Capacity Match* baseline for building *25p_mat*.

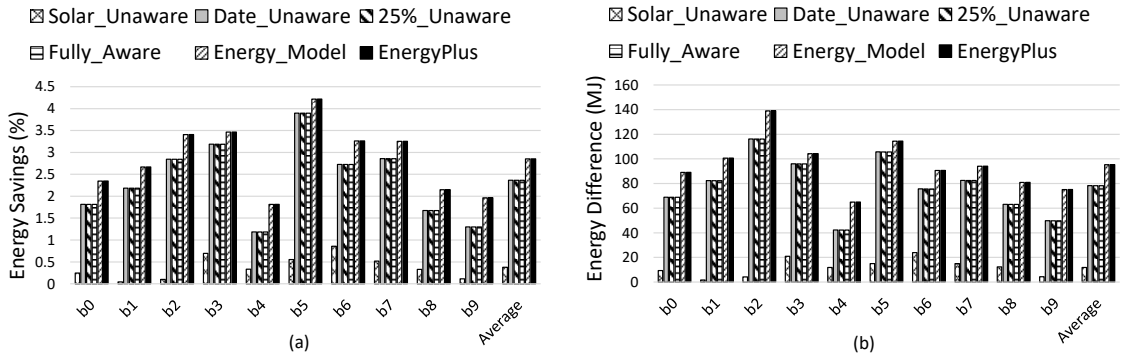


Figure 4.19: (a) Percentage and (b) absolute energy difference for meeting benchmarks with respect to the *Capacity Match* baseline for building *5p*.

Meeting Benchmarks

Figures 4.19, 4.20, 4.21 and 4.22 show the energy savings and energy difference for the meeting benchmarks (Tables 4.8 and 4.9) for our four buildings. On average, the *Energy_Model* achieves the full savings of *EnergyPlus* for *5p*, and 98.2% and 99.4% of the savings for *25p_1win* and *25p_mat*, respectively. For *25p*, the *Energy_Model* and *EnergyPlus* achieve less than 1% savings.

Our prediction model *Date_Unaware* that is trained on the days that are not part of the training set, achieves 83% of the *EnergyPlus* savings for *5p*, 64.7% for

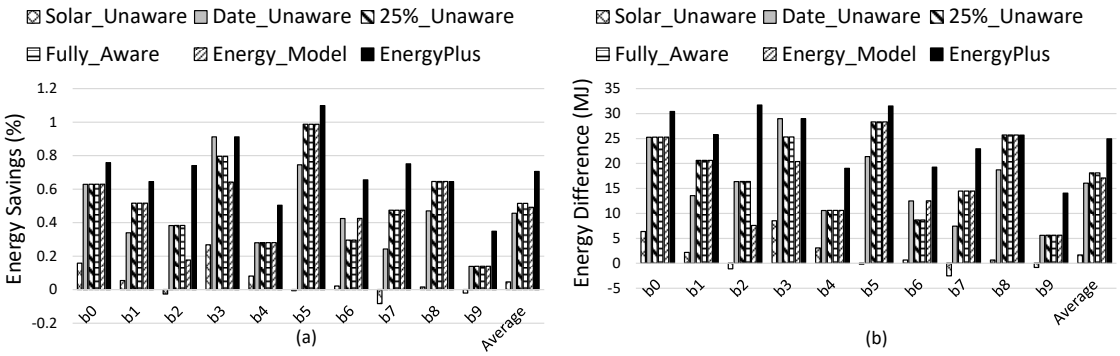


Figure 4.20: (a) Percentage and (b) absolute energy difference for meeting benchmarks with respect to the *Capacity Match* baseline for building *25p*.

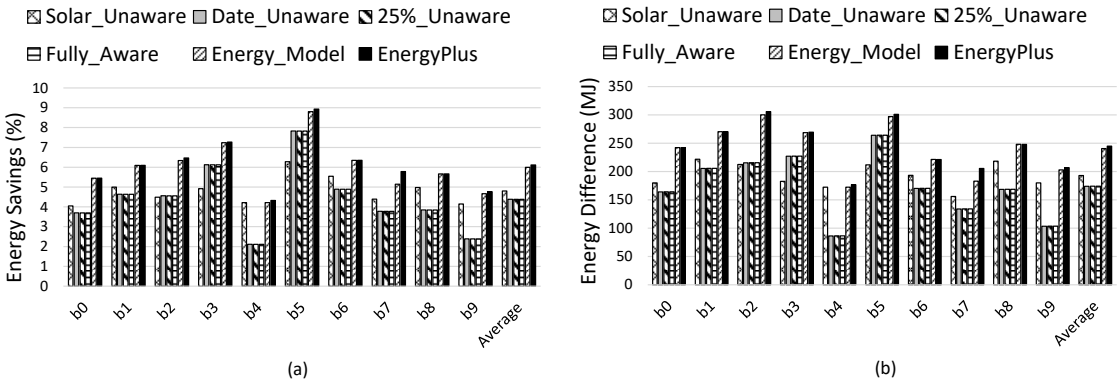


Figure 4.21: (a) Percentage and (b) absolute energy difference for meeting benchmarks with respect to the *Capacity Match* baseline for building *25p_1win*.

25p, 72% for *25p_1win* and 41.3% for *25p_mat*. For *25p*, *Date_Unaware* achieves 0.46% savings in comparison to 0.7% savings for *EnergyPlus*. For *25p_mat*, *Date_Unaware* achieves 0.7% savings, while *EnergyPlus* reduces energy by 1.5%.

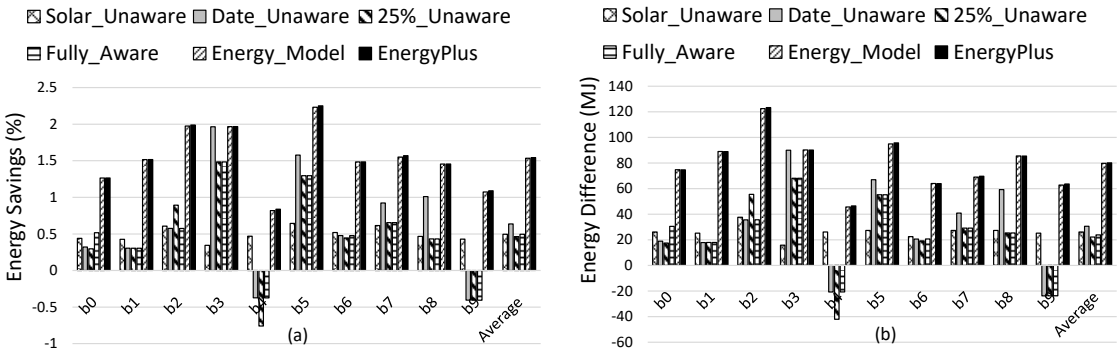


Figure 4.22: (a) Percentage and (b) absolute energy difference for meeting benchmarks with respect to the *Capacity Match* baseline for building *25p_mat*.

This is because *Date_Unaware* achieves less energy savings for benchmarks *b4* and *b9* (Figure 4.22) by making sub-optimal assignments on January 9^{th}, January 26^{th}, February 1^{st} (evident from the previous results of microbenchmarks), and on March 20^{th}. On March 20^{th}, a shorter time during the afternoon seems to be promising for using the south-facing room. However, our meetings are longer than that, and our binary predictors are incapable of handling such situations. *Energy_Model*, however, successfully captures such situations because of the real-valued α.

Other prediction models, such as *Fully_Aware* achieves 83%, 73%, 72% and 30% of the *EnergyPlus* savings, while *25%_Unaware* reaches 83%, 73%, 72% and 32.2% of the maximum savings for *5p*, *25p*, *25p_1win*, *25p_mat*, respectively. For *25p*, both *Fully_Aware* and *25%_Unaware* achieve 0.5% savings compared to the 0.7% savings of *EnergyPlus*. For *25p_mat*, *Fully_Aware* and *25%_Unaware* achieve 0.5% savings, whereas *EnergyPlus* saves 1.5% energy. The reasons for the losses are similar to *Date_Unaware*.

The model *Solar_Unaware*, on the other hand, achieves only 13.3%, 6.5%, 78% and 32% of the *EnergyPlus* savings for *5p*, *25p*, *25p_1win*, *25p_mat*, respectively. We see such high energy savings for *25p_1win* because *Solar_Unaware* excels for *b4*, *b7*, *b8* and *b9*, as shown in Figure 4.21, especially for March 1^{st} and March 4^{th} (evident from Figures 4.9, 4.13 and 4.17), and for March 20^{th}. All of our randomly created benchmarks have these days filled up, which benefits *Solar_Unaware* for building *25p_1win*.

4.6.3 Summary

To summarize, this chapter yields the following key insights:

- Solar factors play an important role in efficiently assigning meetings to appropriate rooms depending on the weather for a given day. Ignoring the solar factors can result in meeting assignments with high energy consumption.
- Our methodology of taking building parameters and solar factors to determine the energy-efficient meeting schedules works effectively across a wide range of meetings and different building characteristics.
- Our energy model based on real-valued solar factor achieves near-oracle energy savings.
- Our energy model using discrete solar factor achieves near-oracle savings for most of the benchmarks, but exhibits small losses for a few due to discretization.
- Buildings with smaller room ratios, fewer windows, and with high-resistance wall material show greater energy savings.

4.7 Conclusion

In this chapter, we extend our meeting room scheduling energy model to include the impact of solar conditions to improve the energy efficiency of meeting room assignment. We propose a methodology that takes the building parameters and solar factors to determine the energy-efficient meeting schedules for a

given day. We include a solar factor in our energy model and identify the conditions impacting this factor. We also develop a methodology to accurately predict the solar factor for a wide range of building models and meeting situations. Our proposed energy model matches the oracle energy savings for a large number of buildings, while our discretized solar model developed using the prediction model achieves up to 88% of the maximum energy savings potential.

CHAPTER 5

DYNAMIC HVAC ENERGY AND OCCUPANT COMFORT OPTIMIZATION

5.1 Introduction

Indoor environmental quality systems, such as heating ventilation and air-conditioning (HVAC) and lighting, consume a majority of the energy within buildings [16]. In contrast to replacing systems, better control algorithms present the most practical means of reducing energy consumption and enhancing Indoor Environmental Quality (IEQ). In current practice, algorithms react to changes in the occupancy while using simple fixed schedules to forecast occupancy. More advanced energy optimization schemes do not consider past discomfort in making future energy optimization decisions. This chapter presents an approach that optimizes energy with respect to past and future discomfort while meeting a cumulative discomfort goal.

Although many predictive occupancy models have been proposed [55, 95, 115, 122, 161], the ramifications of prediction inaccuracy on future control decisions are not well addressed, especially for systems operating in real time. Furthermore, model predictive control (MPC) has been used to constrain instantaneous discomfort while minimizing energy [22, 46, 60, 62, 64, 79, 82, 100, 117, 119, 142]. Since discomfort is felt over time, constraining instantaneous discomfort does not include occupants' memory of discomfort, which reduces the energy savings and perceived occupancy satisfaction. Such systems also fail to dynamically adapt their future energy optimal decisions based on its past discomfort performance.

In this chapter, we propose an MPC-based algorithm that adaptively balances energy and comfort while the system is in operation. Our method uses the history of occupant discomfort in combination with occupancy prediction to constrain the expected discomfort to an allowed budget. In principle, our method saves energy by dynamically shifting discomfort over time based on its real time performance in a similar spirit to MPC that shifts loads for economical energy consumption [159, 179]. The system adapts its behavior according to the past discomfort, and thus plays the dual role of saving energy when discomfort is smaller than the target budget, and maintaining comfort when the discomfort margin is small. If the accumulated past discomfort exceeds the allowed limit due to occupancy mispredictions, the algorithm automatically corrects the situation in real time while still attempting to optimize for energy.

We evaluate our approach using several synthetic occupancy benchmarks and real occupancy datasets in comparison to three baselines: 1) an energy-efficient reactive scheme that heats up the room whenever the room becomes occupied, 2) a smart reactive algorithm that reacts to the occupancy changes while using a simple fixed schedule to forecast occupancy, and 3) an oracle scheme with perfect knowledge of future occupancy. For predictable occupancy patterns, our algorithm operates close to the perfect prediction scheme with 4-10% energy savings over the smart reactive policy while meeting the allowed discomfort budget. For the irregular occupancy patterns, our method meets the discomfort goal while consuming only 2% more energy than the smart reactive policy.

Table 5.1: Key variable definitions considered in this chapter.

Variables	Definitions
E	Instantaneous energy
Occ	Occupancy
d	Discomfort density
$MADD$	Moving Average Discomfort Density
Φ_{MD}	Maximum allowed average discomfort density
T_s	Set temperature
T_r	Room temperature
T_o	Outdoor temperature
T_g	Ground temperature
T_{upper}	Upper set temperature
T_{lower}	Lower set temperature
T_{int}	Intermediate set temperature
T_h	$\lvert T_{upper} - T_r \rvert$ at which $d = 1$, when occupied
T_l	Smallest $\lvert T_{upper} - T_r \rvert$ when occupied, below which $d = 0$

5.2 Problem Statement

5.2.1 Overall Approach

The overall objective of our proposed approach is to minimize cumulative energy consumption (E) while meeting a maximum discomfort goal over a sliding window of fixed length timesteps. Throughout this chapter, we refer to the variables presented in Table 5.1. We consider the heating season, and assume that the controller must set the set temperature, T_s, to a specified comfortable value, T_{upper}, for every timestep that the room is occupied. To save energy, the controller may use a lower specified set temperature, T_{lower}, when the room is unoccupied[1]. The room temperature (T_r) dynamics are a function of the current T_r, ground temperature (T_g), outdoor temperature (T_o) and the current set temperature (T_s). The discomfort at any given timestep is calculated according to the difference $T_{upper} - T_r$ using the model defined later.

[1]One of our baseline schemes also uses an intermediate temperature, T_{int}.

At each timestep, the system calculates a Moving Average Discomfort Density (*MADD*) over the last M occupied timesteps. It tries to maximize energy savings while ensuring that the *MADD* does not exceed a maximum discomfort goal (Φ_{MD}). Intuitively, at any given timestep, the aggressiveness by which the system attempts to save energy in future timesteps depends on recent discomfort as well as future occupancy probabilities. If past discomfort is low and future occupancy is projected to also be low, then the system may try to aggressively reduce energy consumption. If discomfort has been high and future occupancy is also expected to be high, then the system will operate conservatively.

5.2.2 Formulation

The objective function and constraints are given by Equation 5.1. Here, we minimize the cumulative HVAC energy consumption over N timesteps while meeting the maximum discomfort constraint at every timestep i.

$$
\begin{aligned}
\min_{\vec{T_s}} \quad & \sum_{i=1}^{N} E(T_{r,i}, T_{o,i}, T_{g,i}, T_{s,i}) \\
\text{subject to} \quad & \\
& \sum_{j \in \mathcal{J}(i)} d(T_{r,j}) \leq \Phi_{MD} \times \sum_{j \in \mathcal{J}(i)} Occ_j \\
& T_{r,i+1} = f(T_{r,i}, T_{o,i}, T_{g,i}, T_{s,i}) \\
& T_{lower} \leq T_{s,i} \leq T_{upper} \\
& T_{s,i} = T_{upper} \quad \text{if } Occ_i = 1 \\
& \qquad\qquad\qquad\qquad\qquad\qquad \forall 1 \leq i \leq N
\end{aligned}
\tag{5.1}
$$

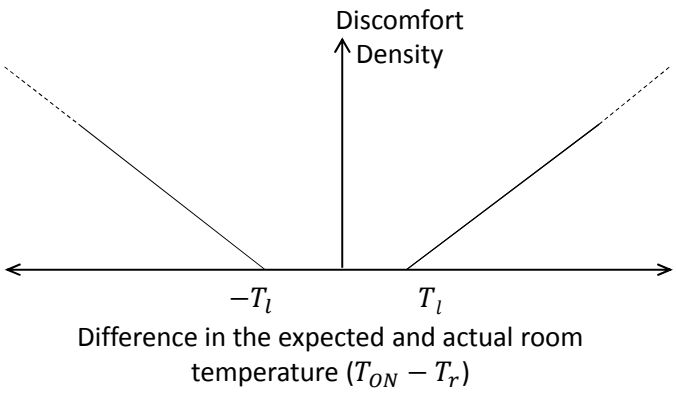


Figure 5.1: Discomfort density.

$\mathcal{J}$ is the set of occupied periods, and $\mathcal{J}(i)$ is the set of occupied indices containing M occupied periods prior to index i. Occ_i at timestep i has a value of 1 when occupied for any time during the timestep and 0 when unoccupied[2].

The discomfort density, $d(T_{r,j})$, is a function of the difference in the expected room temperature (T_{upper}) and actual room temperature (T_r) when the room is *occupied*. Our discomfort model is inspired by the work of Putta et al. [132], and from similar violation-based discomfort models used in the past [46, 64, 82, 100, 142]. Our discomfort model is shown in Figure 5.1 and is formulated according to Equation 5.2. Whenever the temperature difference is less than T_l, the discomfort density is zero; otherwise, it scales linearly. The discomfort probability is highest when the difference is at least T_h. At the temperature difference of T_h, the discomfort density is one.

$$d(T_{r,i}) \triangleq \begin{cases} 0, & \text{if } |T_{upper} - T_{r,i}| \leq T_l \\ \frac{(|T_{upper} - T_{r,i}| - T_l)}{(T_h - T_l)}, & \text{otherwise} \end{cases} \tag{5.2}$$

[2]As discussed later, our system optimizes over a horizon of past timesteps of known occupancies (for which Occ_i is 1 or 0) and future timesteps of unknown occupancies. For these future timesteps, we use expected occupancy values for Occ_i instead of 1 or 0 values.

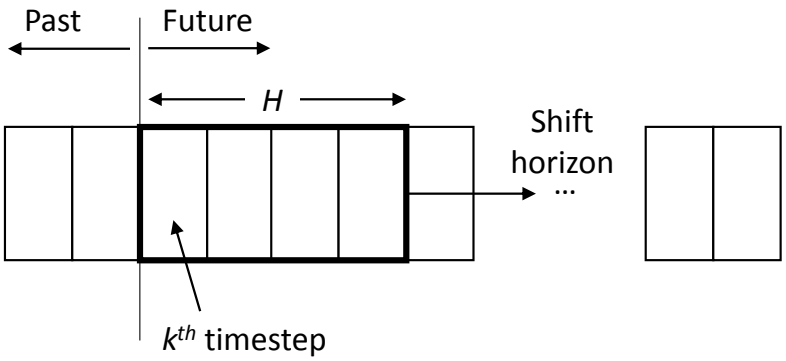


Figure 5.2: Overview of the MPC process.

5.3 MPC-based System Architecture

Model predictive control (MPC) is a technique used in various application domains [88, 109, 157]. It uses a dynamic process model to proactively optimize for the current timestep by anticipating future events. An overview of the MPC process is shown in Figure 5.2. At each timestep k, MPC optimizes for a future horizon of H timesteps. By doing so, it captures the future events that may affect the optimal operation of the k^{th} timestep. After running the optimization, MPC applies the decision to the current k^{th} timestep. Then, for the next $(k+1)^{th}$ timestep, the horizon shifts one timestep and the algorithm optimizes over the next H timesteps. A larger H requires more computation overhead but could lead to a better solution. While MPC with imperfect system models does not guarantee global optimality, using it with feedback from prior MPC decisions helps it achieve promising results.

Figure 5.3 shows the overall system architecture, which consists of our supervisory Predictive Control coupled to conventional HVAC Control. The Predictive Control attempts to produce an optimum T_s for the HVAC Control. The optimum T_s is the reference input for the HVAC Control to maintain the thermal state of the building. We describe these modules in detail below.

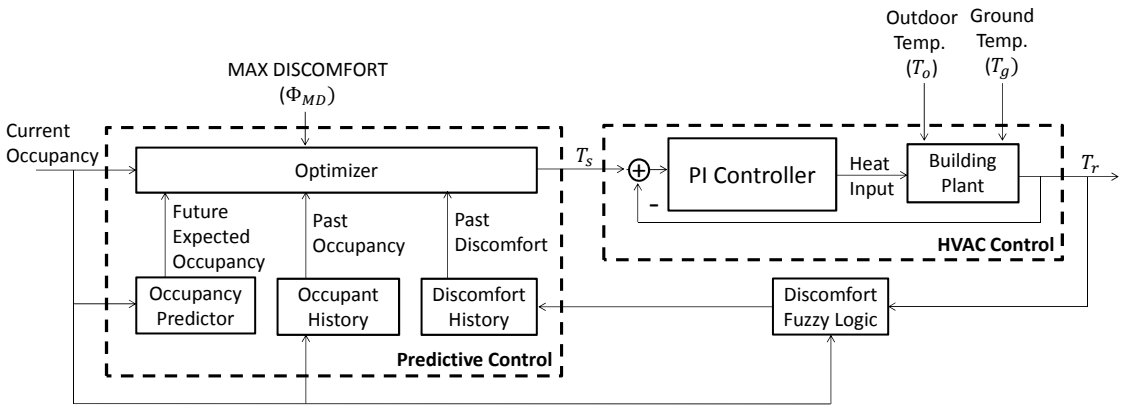


Figure 5.3: MPC-based system architecture.

5.3.1 HVAC Control

The thermal state of the building zone is maintained by a PI controller. Based on the difference between T_s and the room temperature T_r, the PI controller generates a heat input to the building plant. This is the heating power injected to the building to meet the reference T_s requirement. The building thermal model is a linear time-invariant state-space dynamical system

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k \\ y_k &= Cx_k \end{aligned} \tag{5.3}$$

Here, x_k is the temperature state vector containing T_r, and the input vector u_k encompasses the outdoor and ground temperature and the injected heating power. The output vector y_k contains the room temperature T_r. The output matrix C appropriately selects T_r from x_k. The system matrix A contains the building thermal model, while the input matrix B contains the building's response to the applied heat input and weather disturbance.

5.3.2 Predictive Control

The Predictive Control runs an optimization algorithm to generate T_s based on the historical discomfort behavior, current and past actual occupancies, expected occupancies for future timesteps, and Φ_{MD}. It comprises an *Occupancy Predictor* to produce future occupancy probabilities and an *Optimizer* to run the optimization.

Occupancy Predictor

The Occupancy Predictor uses the past occupancy to predict the expected occupancy at each given timestep in the horizon. The prediction is done on a timestep-by-timestep basis, with the expectation at each timestep computed from past occupancy data from days of the week for which similar occupancy patterns are likely.

For offices and labs, we use two separate occupancy models: one for weekdays and the other for weekends. The sample interval is 15 minutes. Thus, a weekday comprises 96 different expected occupancies. Each of these is calculated as an average of some number of past occupancies, with 1 representing an occupied timestep and 0 an unoccupied one. A weekend day has 96 similarly derived timestep expected occupancies. For meeting rooms, we average data over individual weekdays, under the assumption that occupancy patterns will differ among days of the week. Thus, for an office or lab, the expected occupancy for Monday at 10am is identical to Tuesday at the same time, while these could differ for a meeting room.

Optimizer

The Optimizer runs the MPC algorithm for the optimal building HVAC control over a prediction horizon H during the system operation. The MPC algorithm generates a sequence of set points $\overrightarrow{T_s}$ based on the current occupancy, future expected occupancies, and previous discomfort and past occupancies. The optimizer then selects the first element of $\overrightarrow{T_s}$ as a reference for HVAC control. The optimizer assumes accurate outside and ground temperature prediction and uses the building state-space model to compute energy and thermal responses.

At the time index k with the optimizer looking over a horizon H, Equation 5.1, can be written as

$$\min_{\overrightarrow{T_s}} \sum_{i=1}^{k+H} E(T_{r,i}, T_{g,i}, T_{o,i}, T_{s,i}) = \sum_{i=1}^{k-1} E(T_{r,i}, T_{g,i}, T_{o,i}, T_{s,i}) + \min_{\overrightarrow{T_s}} \sum_{i=k}^{k+H} E(T_{r,i}, T_{g,i}, T_{o,i}, T_{s,i}) \tag{5.4}$$

The optimizer cannot affect the past energy, but can attempt to minimize the cumulative future energy over the horizon k to $k + H$. When the room is occupied, the optimizer does not invoke MPC but simply forces $T_{s,k} = T_{upper}$. However, when $Occ_k = 0$, the optimizer attempts to minimize the cumulative energy while keeping the $MADD$ less than Φ_{MD} at every time index h between k and $k + H$. Equation 5.5 shows the optimization problem, where $\mathcal{J}(k + h)$ is the set of occupancy indices with M total occupied periods split between the expected

occupied periods from $k+1$ to $k+h$ and the past occupancy before time k.

$$\begin{aligned} \min_{\vec{T_s}} \quad & \sum_{i=k}^{k+H} E(T_{r,i}, T_{o,i}, T_{g,i}, T_{s,i}) \\ \text{subject to} \quad & \\ & \sum_{j \in \mathcal{J}(k+h)} \mathbb{E}[d(T_{r,j})] \leq \Phi_{MD} \times \sum_{j \in \mathcal{J}(k+h)} \mathbb{E}[Occ_j] \\ & \forall 1 \leq h \leq H \end{aligned} \tag{5.5}$$

The left-hand side of Equation 5.5 is the sum of the expected discomfort, which includes actual past discomfort and the predicted future discomfort over the horizon. The right-hand side is the product of Φ_{MD} and the sum of the expected occupancy, which includes the actual past occupancy and the future expected occupancy over the horizon. If the *MADD* at time index k is very close to Φ_{MD}, the right-hand side is tightened and the optimizer has little room to minimize energy. However, if the gap between the current *MADD* and Φ_{MD} is large, the right-hand side is relaxed, giving more opportunity for energy minimization. If the inequality is not met, the optimization becomes infeasible and the optimizer sets $T_{s,k} = T_{upper}$.

5.4 Methodology

We use the simulation parameters shown in Table 5.2 and the building model of Figure 5.4. We construct the building model using Google Sketchup [11] using realistic materials: a brick exterior, foam-insulated roofing, an insulated concrete slab floor, and double-pane windows. The building model is then converted directly from the CAD geometry and material data to a resistor-capacitor (RC) network using the Sustain framework [66].

Table 5.2: Simulation parameters.

Parameters	**Values**
T_l	±2°C
T_h	±6°C
T_{upper}	21°C
T_{int}	19°C
T_{lower}	15.6°C
K_p	900
K_i	750
Weather	Winter (January)
Location	Elmira, NY
Simulation Timestep	15 minutes
Horizon Length (H)	4 timesteps
Φ_{MD}	10%
Past Occupied Period (M)	40 timesteps

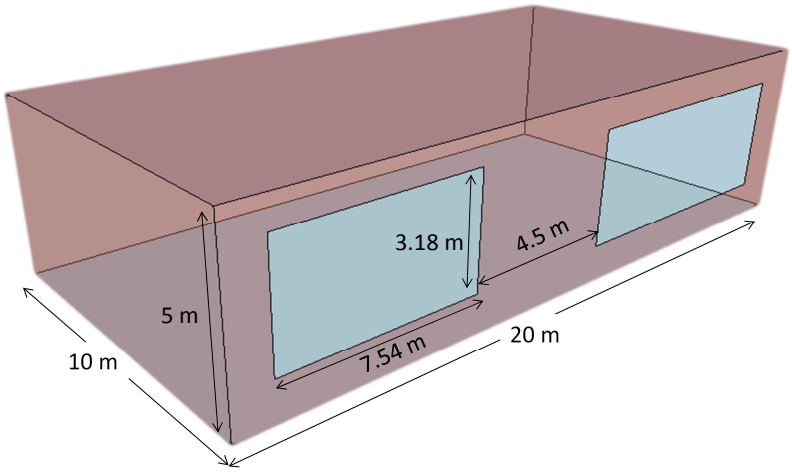


Figure 5.4: Single-zone building model.

Sustain generates a 41-state model encompassing convective and conductive transfer and assumes that the interior air volume is well-mixed. The model does not include radiation. The exterior walls and floor slab are tied to ambient air and ground temperatures which, during simulation, are obtained from an EnergyPlus weather file.

Table 5.3: Synthetic occupancy benchmarks.

Benchmarks	**Periods of Potential Occupancy**
No Occupancy	None
One Hour	9:00-10:00, 13:00-14:00, 16:00-17:00, 18:00-19:00
Two Hours	8:00-10:00, 11:00-13:00, 14:00-16:00, 17:00-19:00
Office	8:00-12:00, 13:00-17:00

5.4.1 Optimization Software

We use CVX [74] to solve the optimization problem of Equation 5.5. The discomfort model is a piecewise-affine function. To mimic the optimization formulation of Equation 5.5, we implement a scalarized multi-objective optimization of energy and discomfort. With discomfort numerically smaller than energy, energy is implicitly optimized with discomfort as a constraint. Therefore, our implementation does not require any scalar parameter to prioritize energy versus discomfort. Furthermore, if CVX is unable to find a feasible solution, the optimizer conservatively sets $T_s = T_{upper}$. This occurs less than 0.3% of the time in our simulations for regular benchmarks and around 10% for irregular data.

5.4.2 Occupancy Benchmarks

We evaluate our approach using real occupancy data as well as synthetic benchmarks (Table 5.3). For the latter, we create time periods of potential occupancy, during which the probability of occupancy is 90% for each timestep. The actual occupancy data includes a Graduate Student Office in Duffield Hall at Cornell University and a lab within the Cornell Nanofabrication Laboratory (CNF). The occupancy data for these spaces are recorded by motion and CO_2 sensors, which we convert to 15 minute timesteps denoting whether or not the room is

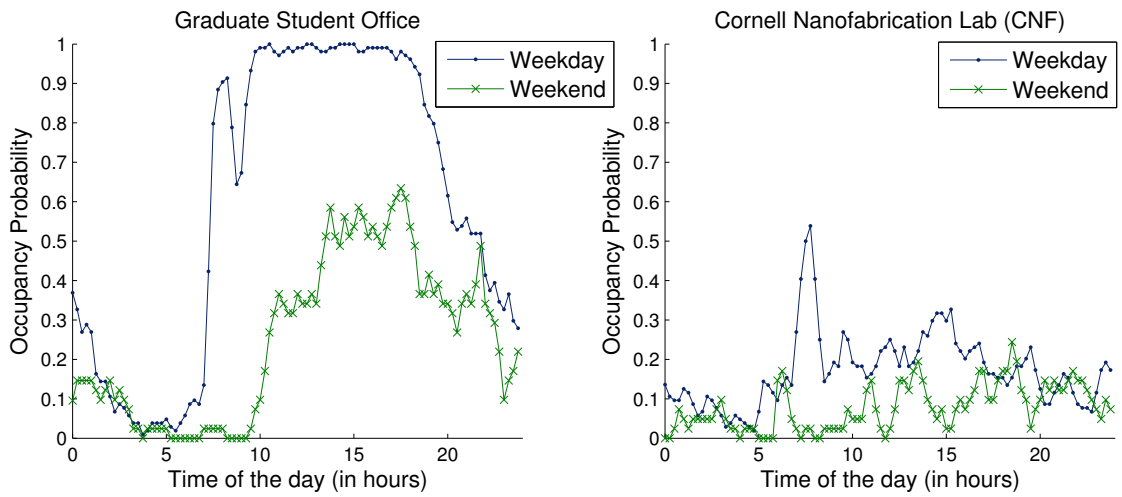


Figure 5.5: Duffield occupancy probabilities.

occupied. We then gather this data for a three month period and calculate a occupancy probability for each timestep on both weekdays and weekends. Our assumption for these spaces is that the expected occupancy at a given time of the day, e.g., 10am, will not significantly differ among different weekdays, but could differ considerably between weekdays and weekends. Figure 5.5 shows the occupancy probabilities of these two spaces over a 24 hour period. The weekday probabilities of the office are high between 7am to 8pm. Weekends are more irregular with occupancy probabilities reaching around 50% during that time period. The CNF occupancy data is far more irregular, which makes optimization more challenging.

We also use occupancy data from the Mitshubishi Electric Research Laboratory 8-North Conference Room [168]. Motion sensors operate asynchronously; we convert these readings to 15 minute occupancy timesteps. However, since this is a conference room, we calculate occupancy probabilities on a weekday basis. That is, we assume that the expected occupancy at a given time of the day could vary significantly among different weekdays, which is borne out by Figure 5.6, which shows the probabilities generated from the MERL data over a

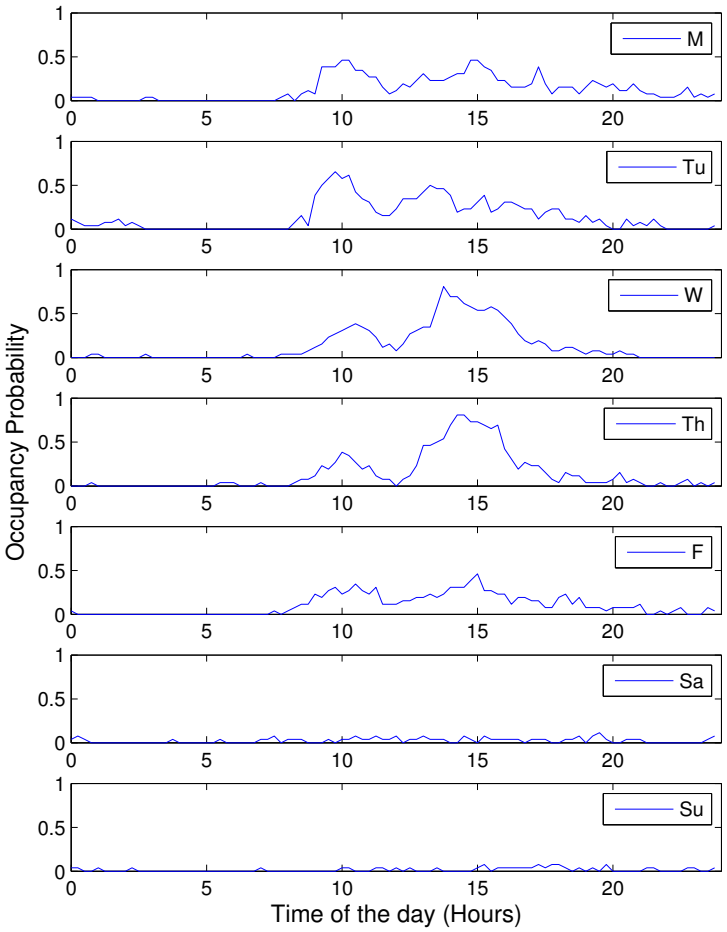


Figure 5.6: MERL occupancy probabilities.

six month period. For instance, the meeting room has a much higher probability of being occupied on Wednesdays and Thursdays compared to Fridays. As expected, weekend probabilities are low.

5.4.3 Baseline Control Schemes

We compare our *predictive* scheme to three baseline control policies. All policies, including those we propose, immediately set the set temperature to T_{upper} whenever the room becomes occupied. The *reactive* policy sets the set temperature to T_{upper} whenever the room is occupied and to T_{lower} when it becomes unoccupied.

As expected, this approach saves energy but at the cost of an unacceptably high *MADD*. To address this shortcoming, the *smart reactive (SR)* policy sets T_s to an interim temperature T_{int} beginning at 6am in anticipation of impending occupancy, and shifts to T_{upper} when occupancy is detected. When the room becomes vacant for 30 minutes, it changes the set temperature back to T_{int}. Beginning at 10pm, SR is like *reactive*. The choice of these specific timings is motivated by a typical office and university occupancy schedule that prioritizes comfort over energy in contrast to the reactive scheme [116]. The SR policy improves *MADD* over *reactive* but at higher energy cost. Finally, the *perfect prediction (PP)* policy is identical to our scheme with the exception of using perfect knowledge of future occupancy (actual occupancy values from our benchmarks) instead of expected occupancies.

5.5 Results

In this section, we present the energy savings and discomfort of our *predictive* scheme compared to the baselines for the real occupancy and the synthetic benchmarks.

5.5.1 Performance of Predictive Scheme

Figure 5.7 illustrates the energy and discomfort of the different control schemes for a representative day using the *Office* benchmark. Beginning at 6am, SR transitions to the intermediate set point T_{int} and then reacts to the first occupant two hours later. After the latest occupancy period, it transitions to T_{int} and even-

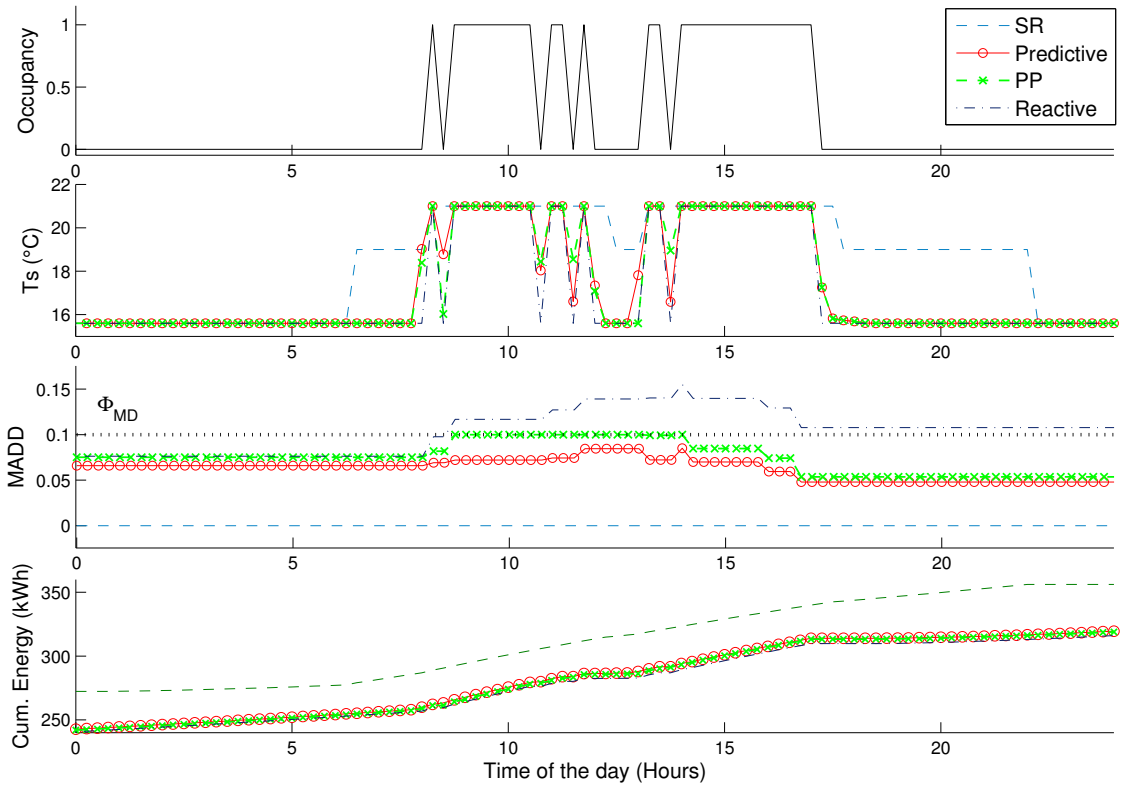


Figure 5.7: Energy and discomfort of different algorithms for *office* occupancy data.

tually to T_{lower}. *Reactive* reacts similarly to the first occupant at 8am, but from the T_{lower} set point, thereby impacting comfort. It reacts in a similarly ineffective manner throughout the day, and often exceeds Φ_{MD}. *Predictive* and *PP* react more smoothly to occupant activities, keeping within Φ_{MD} by proactively conditioning the room before an occupant arrives, and transitioning to T_{lower} during periods of unoccupancy. The cumulative energy consumption over the course of the day of *predictive* is 10.3% lower than SR, and is within 0.2% of PP. The energy of *predictive* is only 1% higher than the *reactive* scheme that frequently violates the discomfort goal.

Figure 5.8(a) shows the percent energy savings over SR for the synthetic and real occupancy benchmarks over a period of 25 days, while Figure 5.8(b) shows

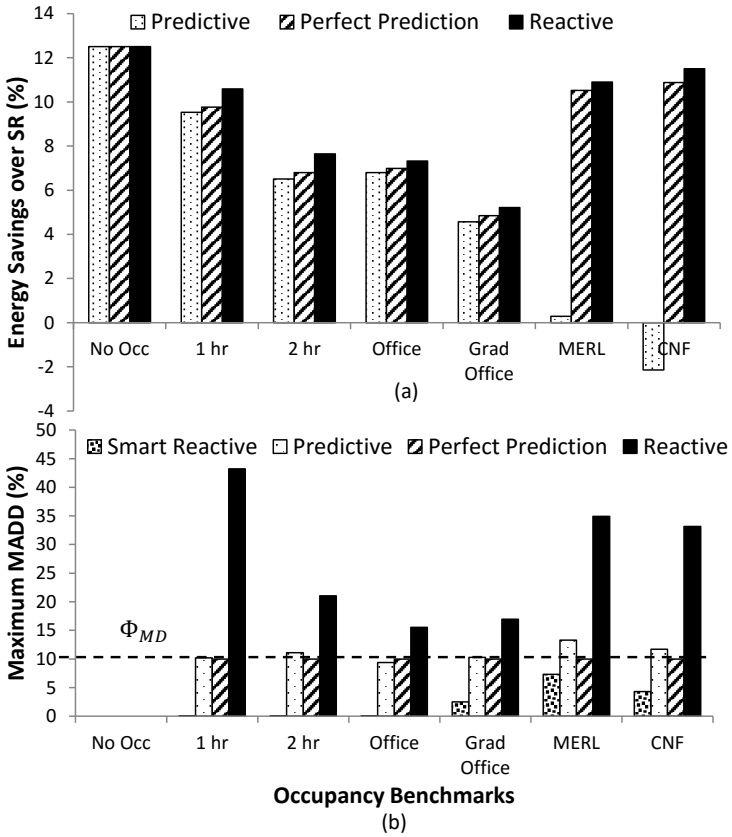


Figure 5.8: Energy and *MADD* for all benchmarks over 25 days.

the maximum *MADD*. The *reactive* scheme has the largest energy savings but its *MADD* often exceeds Φ_{MD}. Over all of the synthetic benchmarks with the exception of *No Occupancy*, the *predictive* scheme achieves 7-10% energy savings, which is almost identical to *PP*. For *Graduate Office*, *predictive* saves 4.5% energy over *SR* and is within 0.3% of *PP*.

The results for *MERL* and *CNF* illustrate the limitations of our approach, in particular our occupancy predictor. These benchmarks lack regularity, with *MERL* being less regular and *CNF* highly irregular. The *predictive* scheme saves only 0.3% energy over *SR* for *MERL*, and expends 2.3% more energy than *SR* for *CNF*. *PP* with its perfect prediction achieves over 10% savings for both benchmarks. Our occupancy prediction averaging scheme at times leads the con-

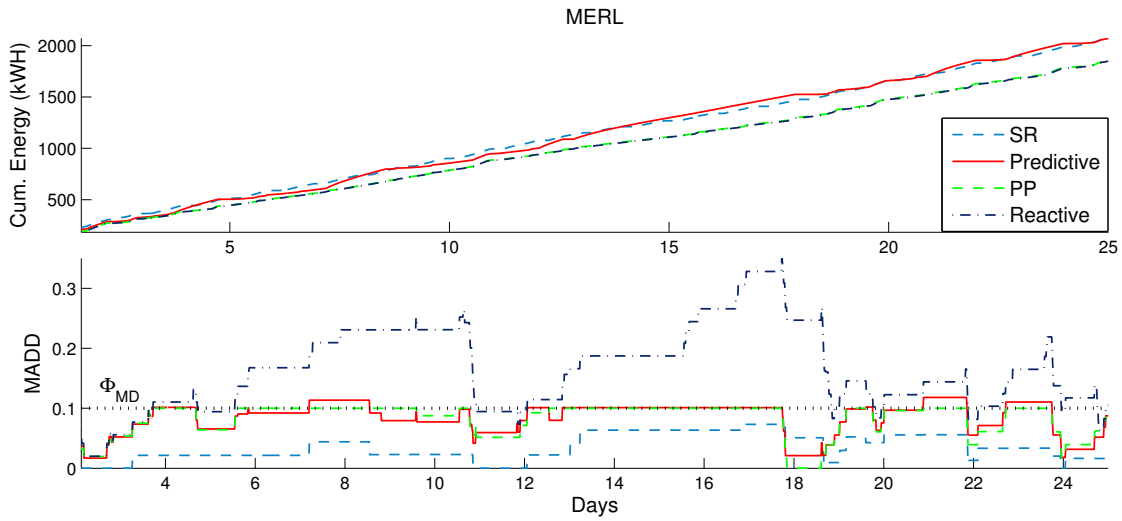


Figure 5.9: Runtime adaptation of energy and discomfort for *MERL*.

troller to anticipate occupancy during unoccupied periods, and to wrongly predict unoccupancy, thereby violating the discomfort constraint. In the latter situation, *predictive* corrects the situation by acting more conservatively in future timesteps, which wastes energy.

This behavior is illustrated in Figure 5.9 for the benchmark *MERL*. When the *MADD* is smaller than Φ_{MD} (on the 11^{th} day for instance), the *predictive* scheme reduces energy consumption. However, due to the irregularity of the benchmark, the *MADD* increases beyond Φ_{MD} on the 13^{th} day. In this situation, the *predictive* scheme automatically corrects the situation and switches to maintaining comfort by conservatively spending more energy to bring the *MADD* below Φ_{MD}. The maximum *MADD* for *MERL* goes up to 0.13 for the *predictive* scheme when simulated for 25 days.

Figure 5.10 shows the adaptive behavior of the *predictive* scheme for the *CNF* benchmark. *CNF* is highly irregular and the *MADD* rarely goes below the Φ_{MD} limit. Thus, the *predictive* scheme rarely saves energy because it must keep the *MADD* below the Φ_{MD} limit, and thereby consumes more energy than the

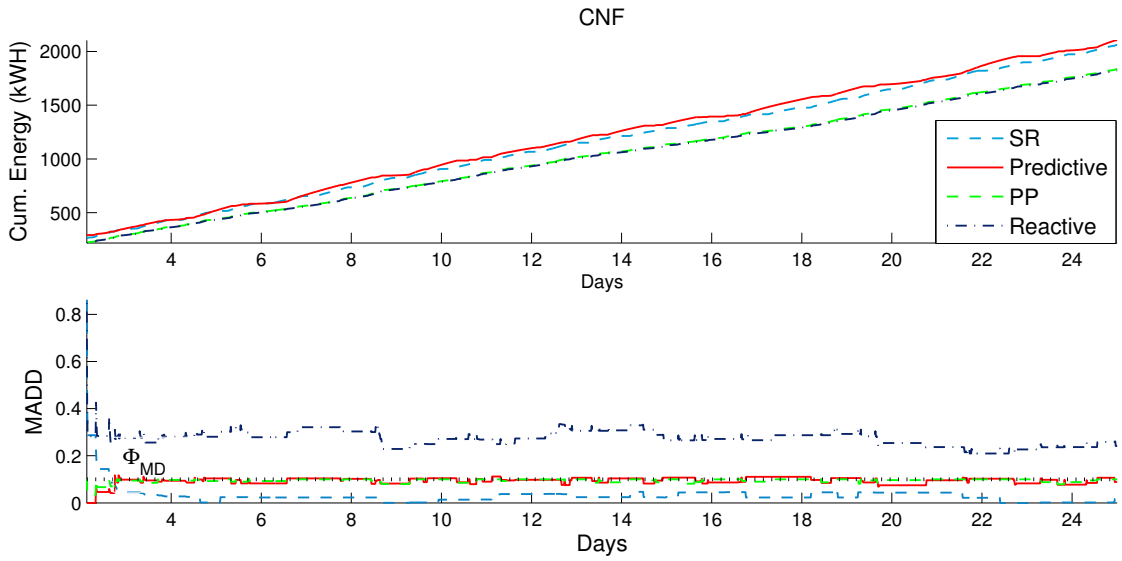


Figure 5.10: Runtime adaptation of energy and discomfort for *CNF*.

baseline *SR*. For *CNF*, the *MADD* reaches as high as 0.12 for *predictive* when simulated for 25 days. These results highlight the need for more accurate occupancy prediction, exploring longer horizon lengths, or a preference for simple rule-based policies such as *SR*, over predictive control policies for irregular occupancy profile.

5.5.2 Daily Performance

Figure 5.11 shows a histogram of the energy savings over *SR* for *Office* and *Graduate Student Office*. For both benchmarks, the daily energy profile of *predictive* closely resembles that of *PP* and *reactive*. For *Graduate Student Office*, all policies–even *PP*–consume more energy (negative energy savings) over *SR* for around 9 days. For these days, the room is occupied from early morning to late night, a near perfect fit to the *SR* schedule. However, there are a few short periods of vacancy for which all schemes try to optimize, resulting in lowering

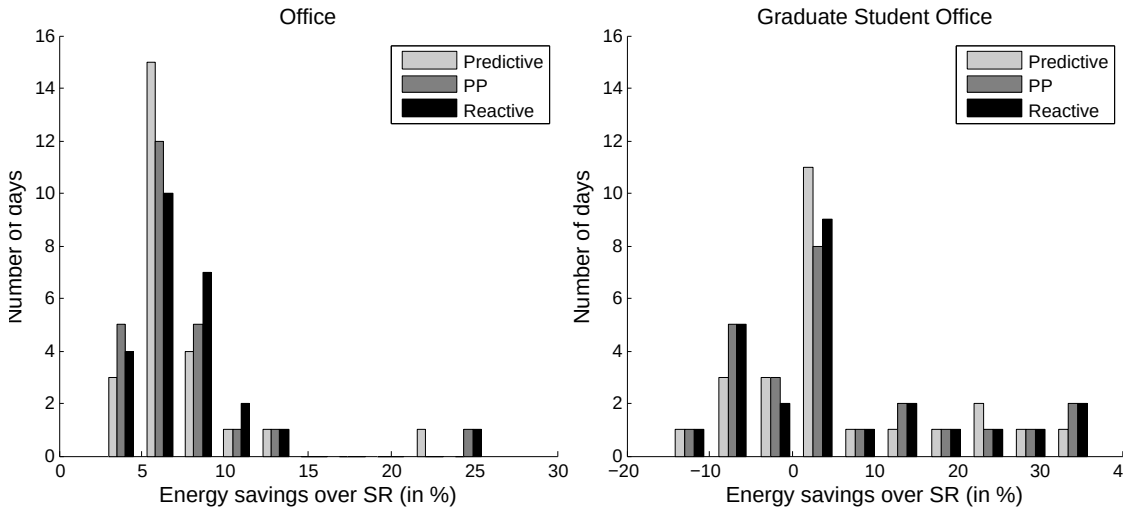


Figure 5.11: Histogram of daily energy savings for *Office* and *Graduate Student Office* benchmarks.

T_s, and thus expending more energy, when the heat must be turned up again. This highlights the need to investigate longer horizons and heuristic solutions for these short vacancy periods.

5.5.3 Effect of Longer Horizon Length

Figure 5.12 shows the effect of increasing the horizon length from 4 to 16 timesteps. As the length of horizon length increases, MPC solves a less local optimization, which improves the energy-comfort gains. Regular benchmark *Grad Office* violates the discomfort constraint for $H = 4$ timesteps. As H increases, MPC spends more energy to operate within the maximum discomfort constraint. Irregular benchmarks *MERL* and *CNF* show both increase in energy savings and reduction in *MADD* as H increases.

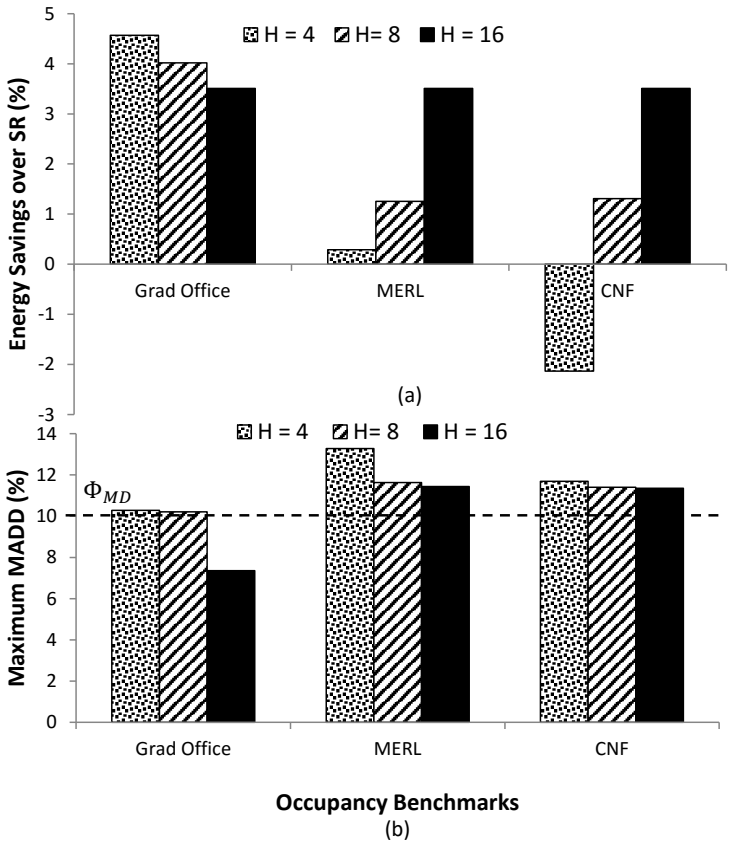


Figure 5.12: Effect of longer horizon length H on energy savings and discomfort compared to the *SR* baseline. Here, H is in timesteps.

5.5.4 Results Without and With Discomfort History

Figures 5.13 and 5.14 compare the energy savings and discomfort achieved *without* and *with* discomfort history. *Without discomfort history* enforces *MADD* to be within Φ_{MD} at every timestep, without carrying forward discomfort from the past. As a result, it fails to dynamically update its discomfort headroom and therefore over-constrains its *MADD*, thereby consuming more energy. *With discomfort history*, however, dynamically updates the discomfort headroom based on the discomfort history, and pushes *MADD* all the way up to the Φ_{MD} limit, resulting in higher energy savings.

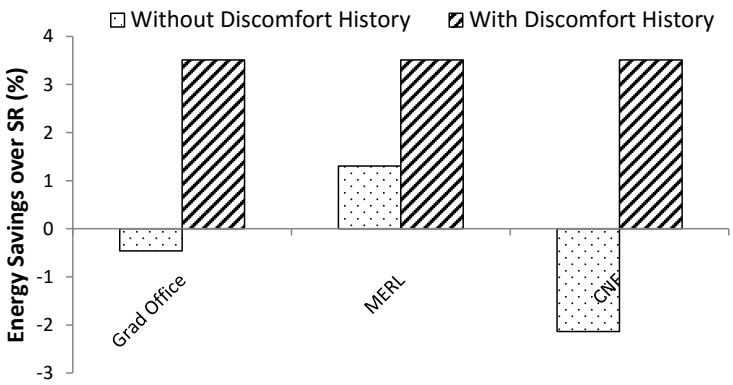


Figure 5.13: Comparison of energy savings *without* and *with* discomfort history for $H = 4$ hours relative to the *SR* baseline.

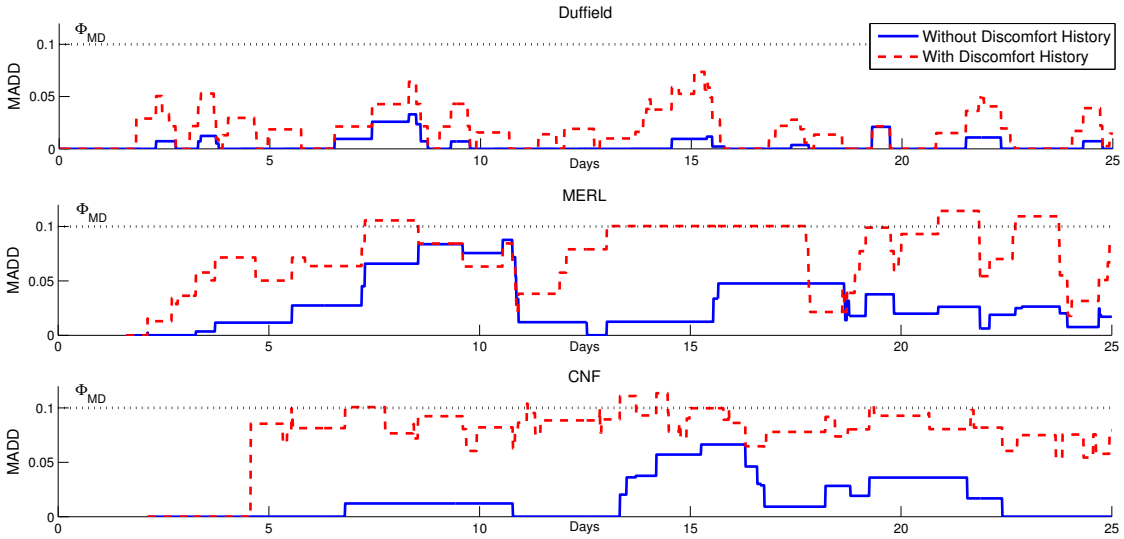


Figure 5.14: Comparison of MADD *without* and *with* discomfort history for $H = 16$ timesteps.

5.6 Conclusion

In this chapter, we present an MPC-based algorithm that uses occupancy prediction and past occupant discomfort to meet a discomfort objective while optimizing energy efficiency. Our system dynamically adjusts how aggressively it attempts to save energy in future timesteps based on recent discomfort as well as expected future occupancy.

Our results show the potential for large energy savings over a smart reactive approach while meeting occupant comfort goals. For regular benchmarks, the MPC scheme achieves large energy savings while violating the discomfort limit marginally. For irregular benchmarks, we observe lower energy savings because our time-based probabilistic learning model does not capture the conditional dependencies within the occupancy data. However, MPC with perfect prediction scheme achieves larger energy savings.

Lastly, increasing the MPC prediction horizon length improves energy savings and occupant comfort for both regular and irregular occupancy data, while ignoring the discomfort history shows lower energy savings.

CHAPTER 6

DYNAMIC GPGPU POWER MANAGEMENT

6.1 Introduction

Dynamic voltage and frequency scaling (DVFS) is a power-saving mechanism that places devices such as CPUs, GPUs, and DRAM channels into lower performance states in order to save power. If these low-power states are used when they will not greatly affect performance, it is possible to save significant energy without unduly impacting performance. Good DVFS policies are vital, since poor decisions can cause both performance and energy losses.

Existing DVFS-based power management techniques, such as AMD's Turbo Core [49, 120] and Intel's Turbo Boost [139, 137, 138], select performance states based on the chip activity seen in the recent past. This may lead to performance and efficiency losses, since this fails to anticipate future performance demands. For instance, lowering the frequency for the next time step may reduce power at the cost of lost performance, while the same action at a future time step may save the same power with no performance loss. Both techniques may equally reduce power, but the latter will yield better energy and performance. This work attempts to alleviate this problem in general-purpose GPU (GPGPU) compute-offload applications.

Previous works that statically optimized individual GPGPU kernels [93, 162], or dynamically optimized over multiple iterations of each kernel [67, 128, 143], ignore future kernel behavior; they utilize information from the last timestep to predict hardware configurations for the next. This falls short for ap-

plications with multiple interleaved kernels with different characteristics and for irregular applications with kernels that vary across iterations [110]. Moreover, these approaches treat each kernel equally in terms of power management decisions, even though kernels may widely vary in their impact on overall application performance. As a result, they may not be able to "catch up" for lost performance or energy savings in later phases with unanticipated behavior, which leads to sub-optimal results. CPU schemes such as phase tracking/prediction only consider the performance of the immediate phase. Similarly, Chen et al. [36] predict the performance of the immediate phase but ignore past behavior.

This chapter presents a GPGPU power management approach that performs inter-kernel optimization while accounting for future kernel behavior. The approach anticipates the expected pattern of future kernels, and their performance and power characteristics, in order to optimize overall application performance and power efficiency. A key component of our approach is *Model Predictive Control* (MPC). MPC optimizes for a future prediction horizon in a receding manner but applies the optimal configuration at the current timestep. However, the implementation overheads of a full MPC algorithm make it unsuitable for the timescales of chip-level dynamic power management, as the problem of maximizing kernel-level energy efficiency under a given performance target is exponential in nature. As such, we propose new greedy and heuristic approximations of MPC that are effective at saving energy with modest performance loss while being applicable to runtime power management. Furthermore, we dynamically adjust the prediction horizon in order to limit the overall performance impact of running our modified MPC algorithm.

To determine the appropriate hardware configuration for a kernel, we develop a prediction model to estimate kernel-level performance and power at different hardware configurations and combine this with a pattern extractor that predicts which kernels will appear in the future. Our overall approach permits MPC to proactively limit performance losses by dynamically expending more energy on high-throughput kernels. This compensates for performance lost when limiting the power in future low-throughput kernels. MPC also avoids spending a disproportionate amount of energy on low-throughput kernels. Instead, it seeks opportunities from the future high-throughput phases to compensate for the performance lost when low-throughput kernels are run at slow DVFS states.

Our approach saves 24.8% energy with a performance loss of 1.8% compared to AMD Turbo Core, and reduces energy by 6.6% while improving performance by 9.6% with respect to state-of-the-art history-based power management schemes.

6.2 Background and Motivation

6.2.1 Heterogeneous Processor Architectures

Modern heterogeneous processors consist of CPU cores integrated on the same die with GPU cores and components such as a Northbridge (NB) and power controllers. Power and thermal budgets may be shared across resources, and some devices (e.g., the GPU and NB) may share voltage rails.

Table 6.1: Software visible CPU, Northbridge, and GPU DVFS states on the AMD A10-7850K.

CPU P States	Voltage (V)	Freq (GHz)
P1	1.325	3.9
P2	1.3125	3.8
P3	1.2625	3.7
P4	1.225	3.5
P5	1.0625	3.0
P6	0.975	2.4
P7	0.8875	1.7

NB P States	Freq (GHz)	Memory Freq (MHz)
NB0	1.8	800
NB1	1.6	800
NB2	1.4	800
NB3	1.1	333

GPU P States	Voltage (V)	Freq (MHz)
DPM0	0.95	351
DPM1	1.05	450
DPM2	1.125	553
DPM3	1.1875	654
DPM4	1.225	720

Table 6.1 shows the different DVFS states for the CPU, NB, and GPU in the AMD A10-7850K processor we study in this work. Changing the NB DVFS impacts memory bandwidth, since each state maps to a specific memory bus frequency. All CPU cores share a power plane. The GPU is on a separate power plane, which it shares with the NB; the NB and GPU frequencies can be set independently, but they share a common voltage.

Lower CPU DVFS states reduce the CPU power and can slightly reduce the GPU power due to a reduction in temperature and leakage. GPU DVFS states change the core frequency of the GPU CUs; however, higher NB states can prevent reducing the GPU's voltage along with the frequency. This can limit the amount of power saved when changing GPU DVFS states. Similarly, if the GPU is at a high-power state, reducing the NB state may only change the NB frequency.

6.2.2 GPGPU Program Phases

A typical breakdown of a GPGPU application is shown in Figure 6.1. The host CPUs first perform some amount of work, shown as *CPU*. After this, they

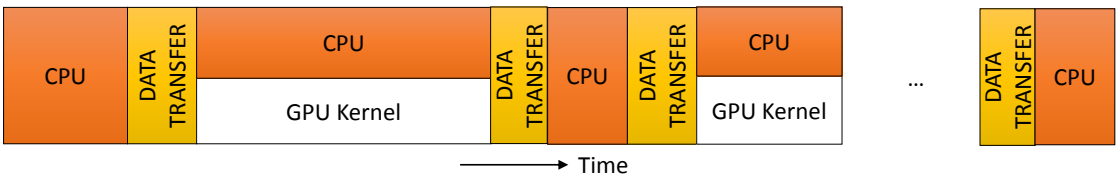


Figure 6.1: Typical GPGPU application phases.

launch computational kernels to the GPUs. A kernel consists of parallel workgroups that are comprised of parallel threads. While the GPU is busy doing computation, the CPU may be waiting for the GPU to finish, preparing data for launching the next GPU kernel, or running parts of the computation concurrently with the GPU.

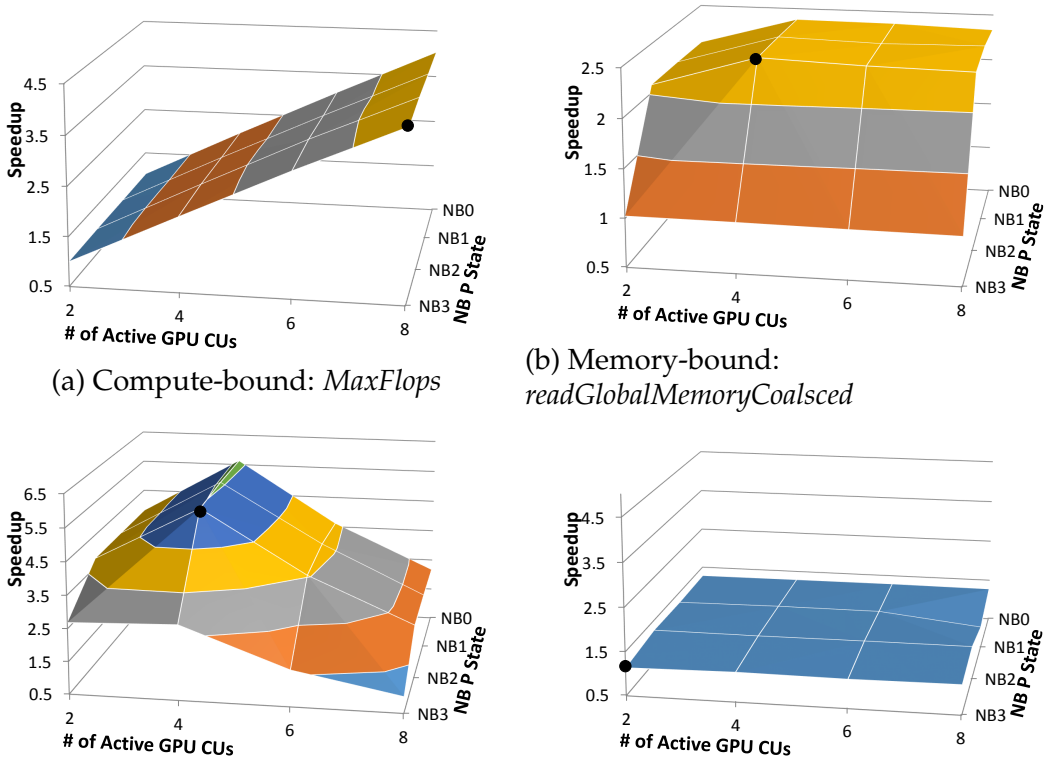
The relative amount of time spent in each phase varies across applications and inputs. For the workloads we investigate in this chapter, the CPU and GPU have little overlapping execution. In Chapter 7, we consider workloads that simultaneously exercise the CPU and GPU.

6.2.3 GPGPU Kernel Characterization

GPGPU kernels show a wide range of performance and energy scaling characteristics and sensitivity to hardware configurations. This section provides a sampling of this behavior to demonstrate the challenges in designing effective power management schemes for GPGPU applications.

Figure 6.2 shows the relative performance of a series of benchmarks as NB DVFS states and the number of active GPU compute units are varied. Each graph also contains a mark at the energy-optimal point.

As can be seen from the figure, these kernels reach their best efficiency at dif-



(a) Compute-bound: *MaxFlops*

(b) Memory-bound: *readGlobalMemoryCoalsced*

(c) Peak: *writeCandidates*

(d) Unscalable: *astar*

Figure 6.2: Performance trends and energy-optimal points of GPGPU kernels at different hardware configurations.

ferent configurations. Compute-bound kernels perform better with more CUs, and their energy-optimal point is at a lower NB state. Memory-bound kernels are sensitive to higher NB states, but the performance saturates from NB2 onwards because NB2 through NB0 have the same DRAM bandwidth. Peak kernels maximize performance and minimize energy at a lower hardware configuration due to destructive shared cache interference [77, 90, 92]. Finally, the performance of unscalable kernels is not affected by hardware changes; these achieve high energy efficiency at the lowest GPU configuration. Similar performance scaling trends have been observed in discrete GPUs [107, 170].

Modern processors attempt to reconfigure their hardware at runtime to best operate on these various applications. State-of-the-practice techniques, such as

AMD's Turbo Core [49, 120] and Intel's Turbo Boost [139, 137, 138], focus on a window of previous kernels or timesteps to inform their decisions. The following sections demonstrate that the lack of future information can often result in poor configuration decisions.

6.2.4 Kernel Runtime Execution Diversity

Table 6.2 shows the execution pattern of the kernels of three benchmarks represented using regular expression. *Spmv* from a modified version of SHOC [65] runs three sparse matrix vector multiplication algorithms ten times each. The *kmeans* application from Rodinia [35] runs the *swap* kernel once, and then iterates the *kmeans* kernel 20 times. The *hybridsort* application from Rodinia runs six different kernels with kernel *mergeSortPass* iterating nine times, each with different input arguments. Each kernel achieves energy optimality at different hardware configurations.

Figure 6.3 shows how the kernel instruction throughput (normalized to the overall throughput) varies during an application's execution. We observe that *Spmv* transitions from high- to low-throughput phases, while *kmeans* demonstrates a low- to high-throughput transition. *Hybridsort* shows multiple phase transitions not only among kernels, but even by the same kernel taking different input arguments. These characteristics are typically seen in irregular applications. For example, graph algorithms vary across input and iteration [110].

Table 6.2: Execution pattern of three irregular benchmarks. Here, A^i indicates kernel A repeats i times. F_1 to F_9 are the invocations of same kernel F, each taking different inputs.

Benchmark	Kernel Execution Pattern
Spmv	$A^{10}B^{10}C^{10}$
kmeans	AB^{20}
hybridsort	$ABCDEF_1F_2F_3F_4F_5F_6F_7F_8F_9G$

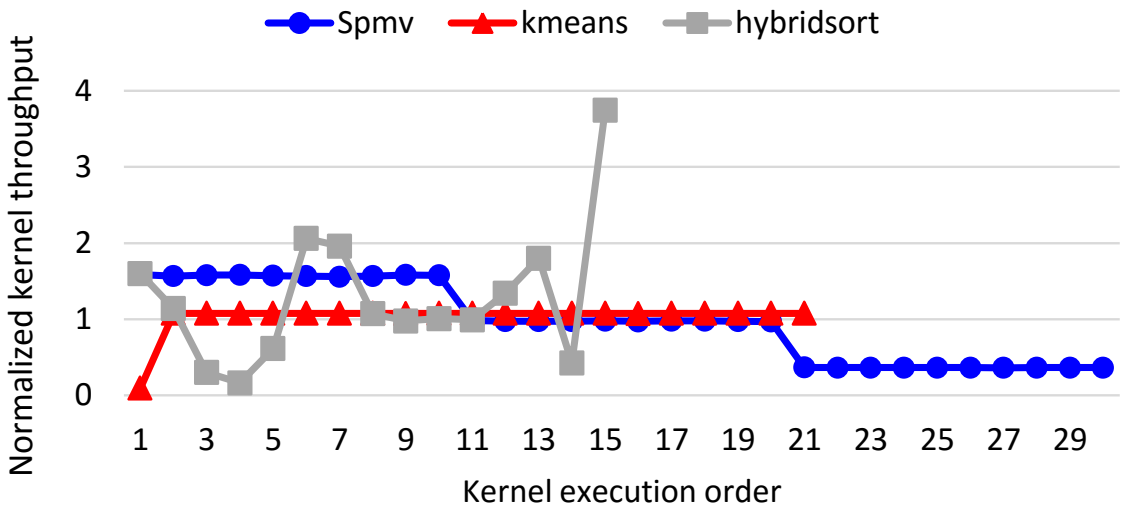


Figure 6.3: Kernel throughput for *Spmv*, *kmeans* and *hybridsort*. Y-axis is normalized to the overall throughput.

6.2.5 Potential of "Future-Based" Schemes

Our goal is to minimize energy while meeting a performance target, in our case, the performance of AMD Turbo Core, which we later explain in Section 6.5.2. In this section, we perform a limit study using two configuration decision algorithms. While both approaches have perfect knowledge of the effect of every hardware configuration on kernel performance and power, the latter also knows the exact pattern of future kernel executions, as well as their performance and power characteristics. Thus, these results could not be obtained in a real system with imperfect predictions.

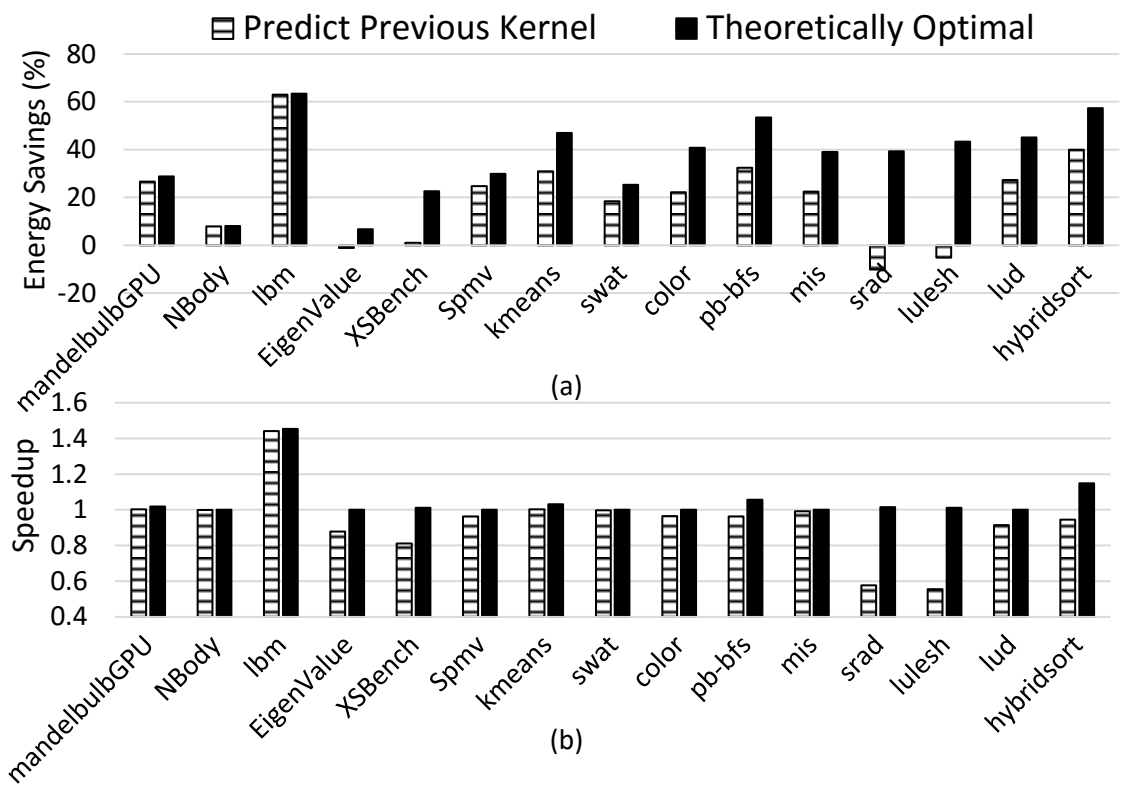


Figure 6.4: Comparison of Predict Previous Kernel and Theoretically Optimal algorithms. (a) indicates energy savings and (b) speedup over AMD Turbo Core.

The *Predict Previous Kernel* (*PPK*) algorithm attempts to minimize energy while assuming the previous kernel will repeat next. It does not look further in the future, but makes its decision based on perfect knowledge of the performance and power characteristics of every hardware configuration with respect to the just completed kernel. PPK represents a best-case scenario for current state-of-the-art history-based algorithms [128, 130, 143], which in practice have errors in their performance and power predictions. In contrast, the *Theoretically Optimal* (TO) algorithm performs a full state space exploration of all future kernels and finds the globally optimal hardware configuration for each kernel iteration.

Figure 6.4 compares the energy and performance of these algorithms against

Turbo Core on the AMD A10-7850K. We observe that PPK matches TO for regular benchmarks such as *mandelbulbGPU*, *NBody* and *lbm*. These benchmarks have a single kernel iterating multiple times; thus, future knowledge is not helpful. However, for the remaining irregular benchmarks, PPK consumes more energy (up to 48%) and degrades performance (up to 46%) compared to TO.

To understand why future knowledge can be so beneficial, we examine the benchmarks *hybridsort*, *Spmv*, and *kmeans* shown in Figure 6.3. In *hybridsort*, all of the kernel invocations differ in throughput, with some varying with inputs. As a result, PPK always mispredicts the next kernel behavior, which leads to far-from-optimal performance and energy results. Applications *XSBench*, *srad* and *lulesh* exhibit similar behavior.

Spmv on the other hand exhibits two high to low throughput transitions. While this behavior results in only two mispredictions by *PPK*, the performance loss compared to the baseline is 4%. This is because PPK reduces the performance of the initial high-throughput phase in order to save energy. On encountering future low-throughput phases, PPK is unable to increase the performance enough to make up for the lost performance; even the highest-powered hardware configuration does not suffice. As such, PPK suffers a performance loss with respect to Turbo Core. The application *lud* shows a similar high to low throughput transition. This result demonstrates the benefits of not only anticipating future kernel patterns, but the performance characteristics of these future kernels as well.

In contrast to *Spmv*, *kmeans* shows a single low to high transition. On encountering the first dominating low-throughput kernel, PPK is temporarily unable to reach the performance target. The performance is degraded so severely that

it cannot be made up in the remaining kernels even when they are run in highest power configuration, thereby consuming more energy. Unaware of the fact that future high-throughput kernels will compensate for the initial low performance, PPK achieves lower energy savings compared to the optimal algorithm. The benchmark *pb-bfs* also exhibits similar results. The *hybridsort* application has multiple high to low changes, and thus suffers both reduced energy savings and performance losses.

Motivated by this fundamental limitation of algorithms like Predict Previous Kernel that ignore the future, and the potential demonstrated by Theoretically Optimal by perfectly predicting it, we propose a future-aware dynamic kernel-level power management policy that anticipates future kernel performance and proactively assigns hardware resources in order to meet its performance and energy targets. We show that a power management policy driven by the principle of feedback and MPC successfully limits the performance loss while improving energy efficiency.

6.3 Problem Formulation

In this section, we formulate the problem and describe algorithms to solve it. The overall objective is to minimize the total kernel-level energy consumption of a GPGPU application without impacting the net kernel performance compared to AMD Turbo Core. In order to compare the performance of a given application over different hardware configurations, we adopt kernel instruction throughput as our performance metric. Equation 6.1 presents the formulation.

$$
\begin{aligned}
&\min_{\overrightarrow{s}} \sum_{i=1}^{N} E_i(s_i) \\
&\text{such that} \\
&\frac{\sum_{i=1}^{N} I_i}{\sum_{i=1}^{N} T_i(s_i)} \geq \frac{I_{total}}{T_{total}} \\
&\text{where } s_i \in S \text{ and } S = \overrightarrow{cpu} \times \overrightarrow{nb} \times \overrightarrow{gpu} \times \overrightarrow{cu}
\end{aligned}
\tag{6.1}
$$

The objective is to minimize the total kernel-level application energy (E) across all N kernels while at least matching the performance of the default Turbo Core algorithm. In Equation 6.1, N is the total number of kernels in an application; and vectors $\overrightarrow{cpu}$, $\overrightarrow{nb}$, and $\overrightarrow{gpu}$ represent the CPU, NB, and GPU DVFS states, while $\overrightarrow{cu}$ represent the ways the number of GPU CUs can be activated. S is the Cartesian product of $\overrightarrow{cpu}$, $\overrightarrow{nb}$, $\overrightarrow{gpu}$ and $\overrightarrow{cu}$. The vector $\overrightarrow{s}$, which belongs to the set S, corresponds to the hardware configurations of N kernels. Each vector element s_i of $\overrightarrow{s}$ represents the hardware configuration for an i^{th} kernel. I_i and T_i are the total number of instructions (thread-count × instruction-count per thread) and execution time of the i^{th} kernel; E_i is the energy consumed by kernel i; and I_{total} and T_{total} are the total number of instructions and the execution time of all kernels in the application in the default Turbo Core approach.

6.3.1 Theoretically Optimal

The previously described Theoretically Optimal (TO) approach assigns a hardware configuration for each kernel instance such that the total kernel energy is minimum with no performance loss. TO solves Equation 6.2, which is the 0-1 Integer Linear Problem (ILP) formulation of Equation 6.1.

$$\min \sum_{i=1}^{N} \sum_{j \in S} E_i(j) X_{ij}$$

such that

$$\sum_{i=1}^{N} I_i - \frac{I_{total}}{T_{total}} \sum_{i=1}^{N} \sum_{j \in S} T_i(j) X_{ij} \geq 0 \tag{6.2}$$

$$\sum_{j \in S} X_{ij} = 1 \qquad \forall\ 1 \leq i \leq N$$

$$X_{ij} \in \{0, 1\} \qquad 1 \leq i \leq N \text{ and } \forall j \in S$$

Here, X_{ij} is a Boolean variable, which is *true* if kernel i is assigned hardware configuration j; $E_i(j)$, and $T_i(j)$ are the associated kernel energy and execution time; and I_i is the instruction count of kernel i. For M possible hardware configurations and N kernels, TO requires $O(M^N)$ searches. TO requires $O(M^N)$ searches. For instance, *Stencil* from Parboil [149] has $N = 100$ kernels. Finding the energy optimal configuration across the $M = 336$ possible configurations would require $\sim 10^{252}$ searches. Searching through so many combinations of kernels and possible hardware configurations makes it impractical to use at runtime.

6.3.2 Predict Previous Kernel

Rather than perform exhaustive search as with TO, current runtime power management approaches optimize the next kernel in execution order based on past knowledge. To reflect this more tractable and runtime feasible approach, we reformulate Equation 6.1 as Equation 6.3.

$$\min_{s_i \in S} E_i(s_i)$$

$$\text{such that} \tag{6.3}$$

$$\frac{\sum_{j=1}^{i} I_j}{\sum_{j=1}^{i} T_j(s_j)} \geq \frac{I_{total}}{T_{total}} \qquad \forall 1 \leq i \leq N \text{ and } \forall s_j \in S$$

Here for every i^{th} kernel, the optimization algorithm chooses the hardware configuration that minimizes the energy of that kernel while ensuring that the total kernel throughput thus far (including this kernel) at least matches that of the default configuration. The polynomial time complexity of $O(M \times N)$ makes the optimization tractable.

The Predict Previous Kernel (PPK) approach described earlier assumes that the last seen kernel or phase repeats again and uses its behavior to estimate the energy optimal configuration of the upcoming kernel. As shown earlier, this approach is far from optimal, which motivates our future-aware MPC approach.

6.4 MPC-based Power Management

In order to develop a future-aware power management scheme, we need to anticipate the sequence of upcoming kernels, and require a model to predict the power and performance of the processor for any GPU kernel. Because of the imperfections associated with any prediction model, we also need to consider feedback to dynamically adjust from past sub-optimal decisions in order to meet the performance target. Our proposed power management scheme applies model predictive control (MPC) to dynamically optimize energy while attempting to meet the performance target.

Recall from Section 5.3 that MPC is a dynamic process control technique that proactively optimizes for a future set of H timesteps and applies the decision for the current timestep. It then shifts the horizon, takes the feedback from past decisions, and re-optimizes for the next H timesteps. In this case, each timestep corresponds to a GPU kernel. A larger H requires more computation overhead but could lead to a better solution. While MPC with imperfect prediction models does not guarantee global optimality, continuous feedback and proactive optimization can compensate for the prediction model inaccuracies.

6.4.1 MPC-Based Online Power Management

Figure 6.5 shows our proposed MPC-based runtime system that attempts to minimize total energy across all kernels while avoiding performance loss. The architecture has four key components: (1) an optimizer, (2) a kernel pattern extractor, (3) a power and performance predictor, and (4) an adaptive prediction horizon generator. This framework runs as a CPU-based software policy between successive GPU kernels.

Optimizer

In theory, MPC minimizes energy while at least meeting the target performance. In our case, we target the performance of the hardware's default power manager. The optimizer runs the MPC algorithm to determine the per-kernel energy optimal hardware configurations (number of GPU CUs; appropriate CPU, GPU, and NB DVFS states) while maintaining the desired performance. It also keeps track of the past performance and instruction counts to determine the available

execution time headroom. This mechanism takes as input estimates from the power and performance model, which we describe later.

Model Predictive Control At each i^{th} step (kernel invocation, in this case), the MPC algorithm optimizes across a window of the next H_i kernels. It determines the minimum energy configuration across those H_i kernels that meets the ongoing performance target and uses that configuration for the current i^{th} kernel. After the execution of that kernel, the prediction window is shifted one kernel in time and the process is repeated for the new window of H_{i+1} kernels. The performance tracker takes the past performance as feedback and dynamically adjusts the execution time headroom for the next optimization. Equation 6.4 shows the MPC formulation for optimizing kernel energy across H_i future kernels under a performance target for an i^{th} kernel.

$$\begin{aligned} &\min_{\vec{s}} \sum_{j=i}^{i+H_i-1} E_j(s_j) \\ &\text{such that} \\ &\frac{\sum_{j=1}^{i+H_i-1} I_j}{\sum_{j=1}^{i+H_i-1} T_j(s_j)} \geq \frac{I_{total}}{T_{total}} \qquad \forall 1 \leq i \leq N \text{ and } \forall s_j \in S \end{aligned} \tag{6.4}$$

MPC Search Heuristic: Traditional MPC approaches use computationally expensive backtracking [33, 99, 164] for each timestep, which is inappropriate for the timescales of dynamic power management. While truly optimizing over multiple H kernels may require backtracking and involves $O\left(N \times (|\overrightarrow{cpu}| \times |\overrightarrow{nb}| \times |\overrightarrow{gpu}| \times |\overrightarrow{cu}|)^H\right)$ searches, we employ greedy and heuristic approximations that permit a polynomial time complexity of $O\left(N \times (|\overrightarrow{cpu}| + |\overrightarrow{nb}| + |\overrightarrow{gpu}| + |\overrightarrow{cu}|) \times H\right)$ to approximate the benefits of backtracking.

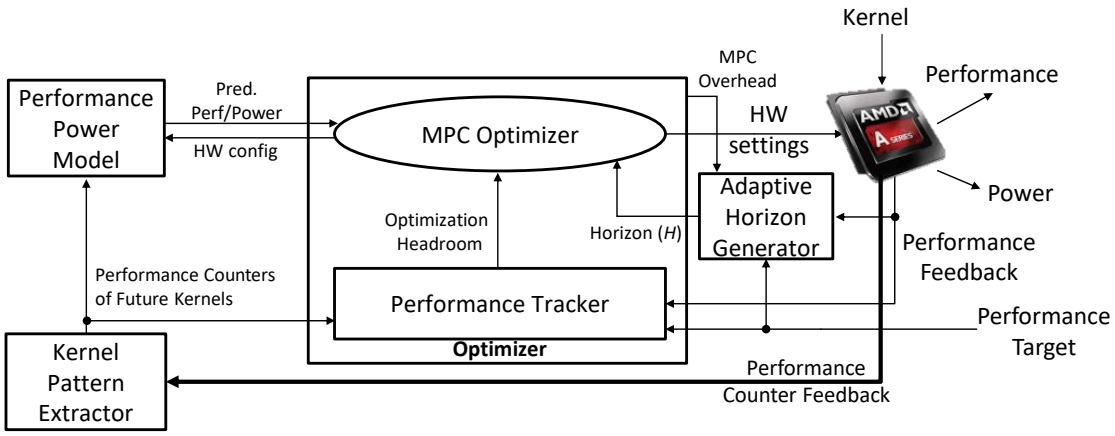


Figure 6.5: MPC-based power management system.

Our approach gathers per-kernel performance information during the first invocation of a GPGPU program in order to minimize the energy of future invocations. Using this information, it determines a search order to optimize the future kernels such that none of the optimized kernels are revisited, thereby reducing the complexity from exponential to polynomial.

The optimization algorithm attempts, in polynomial time, to address two shortcoming of previous approaches such as PPK:

1. The inability to foresee future lower throughput kernels, which may reduce performance due to the inability to "catch up" performance-wise for aggressively saving energy in earlier, high-throughput, kernels; and
2. The inability to foresee future higher throughput kernels, which may reduce energy savings due to the inability to compensate for overly aggressive performance settings in earlier, low-throughput, kernels.

At the conclusion of the execution of each kernel, our approach notes whether the accumulated application throughput is above the overall target

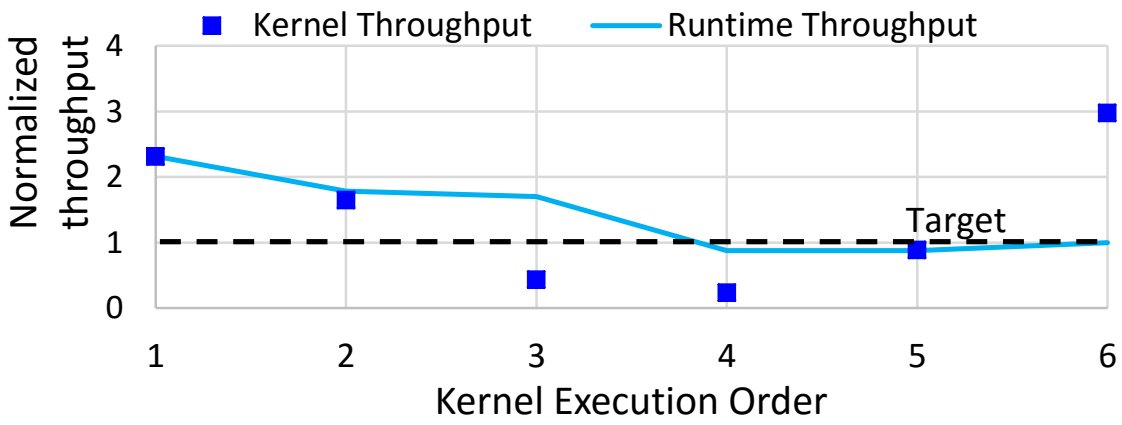


Figure 6.6: An example showing the kernel throughout (squares) and overall application throughput (solid line) during the execution of a hypothetical irregular application. The y axis is normalized to the overall target throughput.

throughput. Those kernels for which the overall throughput is above the target are grouped into the *above-target* cluster and those remaining grouped as *below-target*. The former group are ordered in increasing order by individual kernel performance, and then the latter group in decreasing order. The union of these two groups forms the search order for the heuristic optimization.

Figure 6.6 shows an example execution of a hypothetical irregular application, with the individual kernel (squares) and accumulated application runtime throughput (solid line) normalized to the overall target throughput. The first three kernels $(1, 2, 3)$ are placed in the above-target group because their accumulated runtime throughput values (solid line) are above the overall target throughput (dashed line), while the remaining $(4, 5, 6)$ are placed in the below-target group. We order the above-target group in increasing throughput order (squares). Hence, the order is $(3, 2, 1)$. The below-target group is ordered in decreasing order; therefore the order is $(6, 5, 4)$. The overall search order becomes $(3, 2, 1, 6, 5, 4)$.

With this order determined, the next time the application is invoked, execu-

tion proceeds as follows:

Kernel 1: The optimization is performed in the order $(3, 2, 1)$. The algorithm first estimates the lowest energy configuration for kernel 3 that at least meets the overall target throughput. Any excess performance headroom carries over to kernel 2, for which the lowest energy configuration is found that meets the new target. Any accumulated excess performance headroom carries over to kernel 1, for which the lowest energy configuration is estimated again. The algorithm anticipates the future drop in throughput, which guards against aggressively reducing kernel 1 performance such that it cannot be "made up" in future low performance kernels 2 and 3.

Kernel 2: The optimization order is $(3, 2)$. The algorithm first finds the lowest energy configuration for kernel 3 that at least meets the overall target throughput, taking into account the overall performance thus far. Any excess performance is carried over to kernel 2, for which the lowest energy configuration is found.

Kernel 3: The optimization order is (3); that is, the optimization only considers the current kernel.

Kernel 4: The optimization order is $(6, 5, 4)$. Since the first three kernels have already executed, they are no longer considered. At this point, the algorithm attempts to foresee future higher-throughput kernels (6 and 5) in order to trade off performance for increased energy savings for kernel 4.

Kernels 5 and 6 are optimized in a similar manner.

Greedy Hill Climbing Optimization: To reduce the search overhead and avoid an exhaustive exploration of all possible hardware configurations, we employ

Algorithm 1 Greedy Hill Climbing Optimization

```
1: opt_order ← SortByEnergySensitivity(cpu, nb, gpu, cu)
2: for opt_index := 1 to M step 1 do
3:     hwKnob ← opt_order[opt_index]
4:     hw_config ← findLowestPowerHWConfig(hwKnob)
5:     max_config ← findHighestPowerHWConfig(hwKnob)
6:     min_energy ← ∞
7:     for hw_config to max_config step 1 do
8:         energy ← predictEnergy(hw_config)
9:         if predictPerf(hw_config) ≥ Perf_default then
10:            if energy < min_energy then
11:                min_energy ← energy
12:                opt_config[hwKnob] ← hw_config
13:            else
14:                break
15:            end if
16:        end if
17:    end for
18: end for
19: if predictPerf(hw_config) ≥ Perf_default then
20:    return opt_config
21: else
22:    return failsafe_config
23: end if
```

greedy hill climbing, as presented in Algorithm 1.

Among the hardware knobs, i.e., the DVFS states (CPU, NB or GPU) and GPU CUs, the algorithm first estimates their energy sensitivities[1] using the prediction model, and sorts them in descending order (line 1). The knob with the highest energy sensitivity is selected first and then its corresponding configuration is searched in a hill-climbing fashion such that the predicted kernel energy continues to decrease while meeting the default performance target (lines 7-17). The search stops once the energy increases (line 14). The optimization then continues with the next highest energy sensitive knob, and so on. In the event that the algorithm fails to meet the overall performance requirements, it defaults to an empirically determined fail-safe configuration of [P7, NB2, DPM4, 8 CUs].

[1]Ratio of predicted change in energy to change in configuration.

While this approach compromises optimality, the number of energy evaluations reduces from ($|\overrightarrow{cpu}| \times |\overrightarrow{nb}| \times |\overrightarrow{gpu}| \times |\overrightarrow{cu}|$) to ($|\overrightarrow{cpu}| + |\overrightarrow{nb}| + |\overrightarrow{gpu}| + |\overrightarrow{cu}|$), or a factor of 19×. The greedy search in conjunction with the MPC heuristic reduces the search cost by 65× compared to an exhaustive MPC search involving backtracking, which makes our approach suitable for runtime optimization.

Performance Tracker The performance tracker dynamically adjusts the execution time headroom for MPC optimization based on the desired performance target, execution history of past kernels, and performance behavior of future kernels. The performance requirement for an i^{th} kernel is enforced according to Equation 6.5.

$$\frac{\sum_{j=1}^{i-1} I_j + \mathbb{E}[I_i]}{\sum_{j=1}^{i-1} T_j(s_j) + \mathbb{E}[T_i(s_i)]} \geq \frac{I_{total}}{T_{total}} \tag{6.5}$$

The headroom for MPC optimization is dynamically adjusted using the net performance of the past $i-1$ kernels and performance counters from the kernel pattern extractor. The expected kernel time $\mathbb{E}[T_i]$, provided by the performance predictor, must be within this updated headroom (Equation 6.6). Significant performance slack provides the optimizer with the opportunity to aggressively save energy. With less headroom, the optimizer operates more conservatively, choosing higher performance, and higher energy, configurations.

$$\mathbb{E}[T_i(s_i)] \leq \left(\sum_{j=1}^{i-1} I_j + \mathbb{E}[I_i]\right) \Big/ \left(\frac{I_{total}}{T_{total}}\right) - \sum_{j=1}^{i-1} T_j(s_j) \tag{6.6}$$

Kernel Pattern Extractor

GPGPU applications commonly execute many kernels in a regular order. As shown later in Table 6.4, several applications present regular execution patterns. There may also be distinct patterns within the same kernel across multiple invocations due to input data set changes. We use these patterns to predict the future behavior of the kernels and to store their performance counters for future use by the optimizer. The mechanism we develop to extract kernel execution patterns is composed of three steps: (1) build the kernel execution list over time; (2) identify the kernel signature; and (3) pass the future kernel information to the optimizer.

The kernel pattern extractor samples the performance counters at runtime and stores it in a reduced format. These performance counters are then used by our power and performance predictor. The execution ordering list is dynamically extracted when our framework first encounters the benchmark. At this initial stage, our MPC framework simply runs PPK while it dynamically extracts the pattern.

The pattern extractor implements the dynamic pattern extractor as proposed by Totoni et al. [156]. It identifies different kernels through their signature, extracts the execution pattern once it observes a repetitive behavior, and stores the ordering along with the performance counters.

To find the kernel signatures at run-time, we first reduce the number of performance counters to reduce the runtime compute and storage overheads. This is done by clustering the counters that are more correlated in a similar fashion as Zhu and Reddi [178]. Based on the clustering, we select eight representa-

Table 6.3: GPU performance counters.

Name	Description
GlobalWorkSize	Global work-time size of the kernel.
MemUnitStalled	Percentage of GPUTime the memory unit is stalled.
CacheHit	Percentage of fetch, write, atomic, and other instructions that hit the data cache.
VFetchInsts	Average number of vector fetch instructions from video memory executed per work-item.
ScratchRegs	Number of scratch registers used.
LDSBankConflict	Percentage of GPUTime LDS is stalled by bank conflicts.
VALUInsts	Average number of vector ALU instructions executed per work-item.
FetchSize	Total kB fetched from video memory.

tive performance counters that reflect any input data and kernel characteristics, as presented in Table 6.3. Our pattern extractor stores eight of these performance counters along with the kernel time and power in double-precision values, which accounts for 80 bytes, for each dissimilar kernel.

Next, we approximate kernels with similar performance by binning their counter values according to the following formula: $\text{bin}_i = \lfloor \log u \rfloor, \forall u \in S$, where S is the eight performance counters. The tuple $(\text{bin}_1, ...\text{bin}_k)$ is the signature.

The kernel signature and the execution ordering together maintain an indexed list of kernels. In successive iterations, the pattern extractor identifies which kernel signature to expect in the future and passes the corresponding performance counters to the prediction model, and the expected instruction count to the optimizer. It also dynamically updates the stored kernel performance counter values based on the performance counter feedback of the last executed kernel.

Performance and Power Predictor

The performance and power predictor uses an offline trained model that predicts the power and performance of a kernel. It takes as inputs the performance counters of future kernels from the kernel pattern extractor and the corresponding hardware configuration, and gives the power and performance estimates of a kernel for any desired hardware configuration.

Our performance and power model uses machine learning to model the behavior of the integrated GPU. We use a Random Forest regression algorithm [29] to capture the GPU power and performance behavior. Random Forest is an ensemble learning method that creates multiple regression trees for each random subset of the training data. The predicted class is the mean prediction from these individual regression trees. We selected Random Forest because it gave the highest accuracy among other learning algorithms.

For the kernel performance and power prediction, Random Forest uses the kernel-level GPU performance counters, kernel execution time, and GPU (including NB) power numbers for several benchmark suites executed under different GPU/NB configurations. Since the GPU and NB share the same voltage plane, the GPU power numbers also capture the NB power and the effect of changing NB configurations. The model is trained offline and the system-level software implements the predictor. The accuracy of this model is described in Section 6.6.4. For CPU power prediction, we use a normalized V^2f model because the CPU usually busy waits while the kernel is executing.

Adaptive Horizon Generator

The choice of a horizon length H is a tradeoff between the quality of the solution and the computation overhead of the algorithm. The overhead may be particularly problematic for applications with short GPU kernels separated by short CPU times. Even with our polynomial time MPC algorithm, the value of H must be carefully chosen to avoid significant runtime overheads for these applications.

To address this issue, we propose to dynamically adapt the value of H on a per-kernel basis at runtime. The adaptive horizon generator determines the horizon length H_i for each upcoming i^{th} kernel such that the total performance loss (the MPC overhead plus the performance loss due to MPC approximations and imperfect predictions) remains bounded.

To determine the horizon H_i for each i^{th} kernel, we make use of the information gathered on the first invocation of the application, namely: (1) the number of kernels N, (2) the average per-kernel horizon length $\mathbb{N}$ calculated from the search order, and (3) the total time to run PPK during the initial invocation T_{PPK}.

The adaptive horizon generator determines a horizon length H_i of the present i^{th} kernel based on the estimated MPC overhead ($H_i \times \frac{\mathbb{N}}{N} \times T_{PPK}$), the total execution times of the previous $i-1$ kernels ($\sum_{j=1}^{i-1} T_j$), the total MPC optimization overhead incurred for the previous $i-1$ kernels ($\sum_{j=1}^{i-1} T_{MPC,j}$), and the estimated execution time of the present i^{th} kernel (T_{total}/N). We attempt to bound the performance penalty relative to the baseline Turbo Core execution time so far, including the current kernel ($i \times T_{total}/N$), to a factor α, as shown below.

$$\frac{H_i \times \frac{\mathbb{N}}{N} \times T_{PPK} + \sum_{j=1}^{i-1}(T_j + T_{MPC,j}) + T_{total}/N}{i \times T_{total}/N} \leq 1 + \alpha$$

Solving for H_i, we get:

$$H_i \leq \frac{N}{\mathbb{N}} \frac{(1 + \alpha - \frac{1}{i})\frac{i \times T_{total}}{N} - \sum_{j=1}^{i-1}(T_j + T_{MPC,j})}{T_{PPK}}$$

We take the floor of H_i to create an integer value, and further bound H_i to be between 0 and N.

6.5 Experimental Methodology

In this chapter, we use an AMD A10-7850K APU as our experimental platform. We use this APU in our studies because, due to its more stringent thermal constraints, it more aggressively manages power compared to discrete GPUs. The core concepts, observations and insights from this work are also applicable to other heterogeneous processors.

We implemented the MPC framework on the host CPU of the AMD A10-7850K APU running at the hardware configuration of [P5, NB0, DPM0 and 2 CUs]. The CPU runs the MPC algorithm between GPU kernel invocations. While in a real implementation, there may be an idle CPU available to run the algorithm during CPU phases between the GPU kernels, we assume a worst-case scenario in which the GPU kernel invocations occur back-to-back, or a CPU is not available to run the algorithm during the CPU phase. In our studies, the horizon length generator attempts to limit the maximum performance loss to an α of 0.05 (5%).

In order to simulate our approach as well as competing schemes, we captured performance and power data on the AMD hardware for 336 APU hardware configurations by varying the CPU, NB and three out of five GPU DVFS states as shown in Table 6.1, and changing the number of active GPU CUs from 2 to 8 in steps of 2. We use AMD CodeXL to capture the runtime GPU performance counters and measure CPU and GPU power from the APU's power management controller at 1ms intervals. The NB power is included in the GPU measurement, since they share the same voltage rail. This extensive power and performance information, which is captured at run-time for the individual kernels for each of the benchmark suites described in the next subsection, permits accurate comparison of the performance and energy use of different power management schemes with respect to the baseline AMD Turbo Core approach.

6.5.1 GPGPU Benchmarks

We study 73 benchmarks from 9 popular benchmark suites and sample 15 of them (Table 6.4) that have wide-ranging behavior and utilize the hardware in different ways. Within the 73 benchmarks we studied, we found that 75% are irregular and 44% of the kernels varied significantly with input. To represent such a distribution, we categorize our benchmarks according to their kernel execution pattern. Regular benchmarks have a single kernel that iterates multiple times; we include these to show that MPC does not degrade performance or energy efficiency for regular applications. Irregular applications are categorized into the ones with repeating and non-repeating kernel patterns, and those which vary with inputs.

Table 6.4: Benchmarks with their execution pattern.

Category	Benchmarks	Benchmark Suite	Regular Expression
Regular	mandelbulbGPU	Phoronix [10]	A^{20}
	Nbody	AMD APP SDK [2]	A^{10}
	lbm	Parboil [149]	A^{10}
Irregular with repeating pattern	EigenValue	AMD APP SDK [2]	$(AB)^5$
	XSBench	Exascale	$(ABC)^2$
Irregular w/ non-repeating pattern	Spmv [65]	SHOC [42]	$A^{10}B^{10}C^{10}$
	kmeans	Rodinia [35]	AB^{20}
Irregular w/ kernels varying with input	swat	OpenDwarfs [57]	No pattern. Multiple iterations of a same kernel varying with input arguments.
	color	Pannotia [34]	
	pb-bfs	Parboil [149]	
	mis	Pannotia [34]	
	srad	Rodinia [35]	
	lulesh	Exascale	
	lud	Rodinia [35]	
	hybridsort	Rodinia [35]	

6.5.2 Baseline Schemes

We report the energy and performance improvements with respect to the default Turbo Core scheme in the AMD A10-7850K [20]. Turbo Core is a state-of-the-practice technique that balances power and performance under thermal constraints. It controls the DVFS states based on the recent resource utilization, and shifts power between the GPU and CPU based on their recent load. For the OpenCL GPGPU applications, the CPU busy-waits while the GPU is executing the kernel. Therefore, Turbo Core does not drop the CPU DVFS states as long as the system stays within its TDP.

We also compare our MPC method to the PPK and TO schemes described in Section 6.3. PPK represents the state-of-the-art predictive techniques for GPGPU benchmarks that do not consider future kernel behavior [128, 130, 143],

while TO is an impractical scheme that demonstrates what is theoretically possible. Furthermore, since the CPU is mostly busy-waiting, due to the nature of the available benchmarks, we also compare the energy savings both with and without the CPU energy to provide a fair assessment.

Upon encountering the benchmark for the first time, all the schemes run PPK, while dynamically extracting the kernel execution pattern. At this stage, our framework starts with no stored knowledge. The very first kernel is run at fail-safe since no performance counters are available to predict its power and performance. Subsequently, PPK uses the last kernel's performance counters to predict the next kernel's energy-optimal configuration.

6.6 Results

In this section, we first show the benefits of MPC after the initial run of the application has been performed, and then explore how the initial energy and performance losses of running PPK the first time are amortized over multiple executions, as encountered in real-world applications. Unless otherwise stated, all of our results include the energy and performance overheads of the MPC and PPK optimizations.

6.6.1 Energy-Performance Gains

Figure 6.7 compares the energy savings and performance impact of PPK and MPC over AMD Turbo Core. MPC fares similarly to PPK for the three regular benchmarks with a single repeating kernel. However, the differences are more

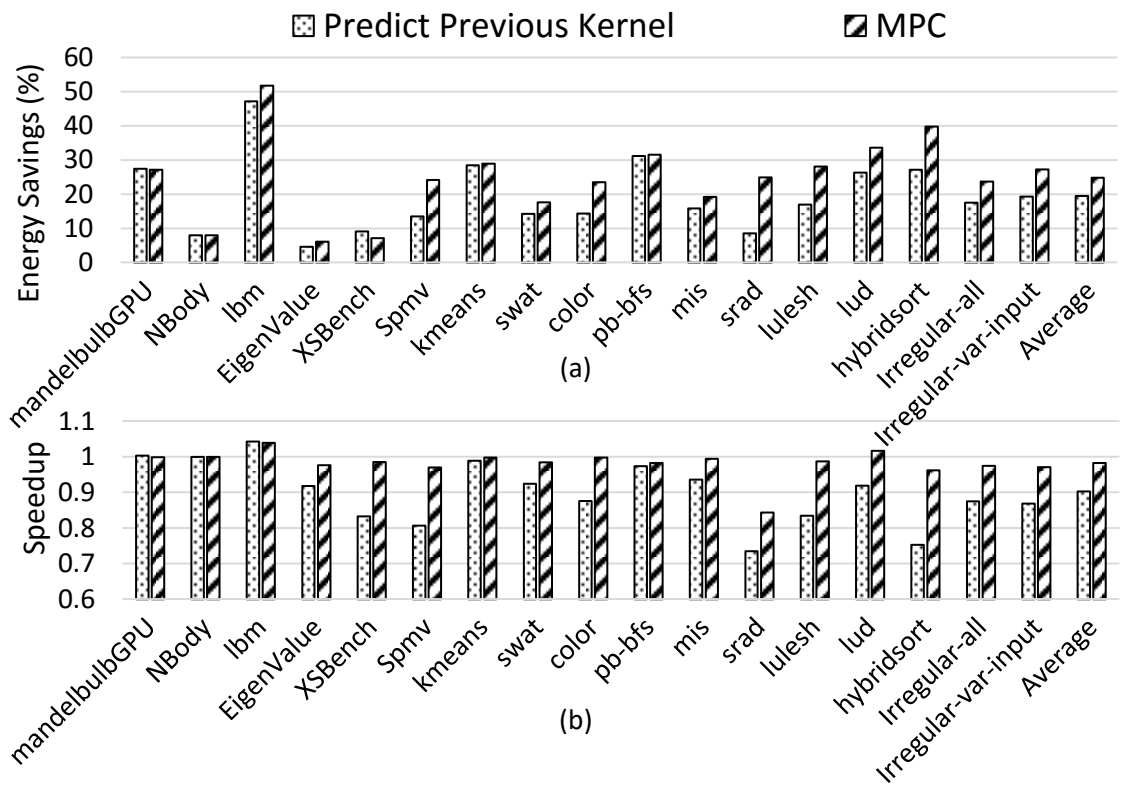


Figure 6.7: PPK and MPC (a) energy savings and (b) speedup over AMD Turbo Core.

pronounced for the irregular benchmarks whose complex patterns benefit from additional future knowledge. Here, MPC considers the future kernel behavior and mitigates the performance losses of looking only a single kernel into the future, while simultaneously saving energy. Overall, including the MPC overheads, MPC achieves a 24.8% energy savings over Turbo Core with a 1.8% performance loss. Except for *srad*, MPC achieves a maximum performance loss of 3.8% for *hybridsort*. This is because MPC adaptively tunes the MPC horizon and restricts the total performance loss to 5%. The 15.7% performance loss for *srad* represents a worst-case scenario for our MPC approach with imperfect prediction. Here, the prediction model mispredicts during the last phases of *srad*, and MPC is unable to recover from the performance loss.

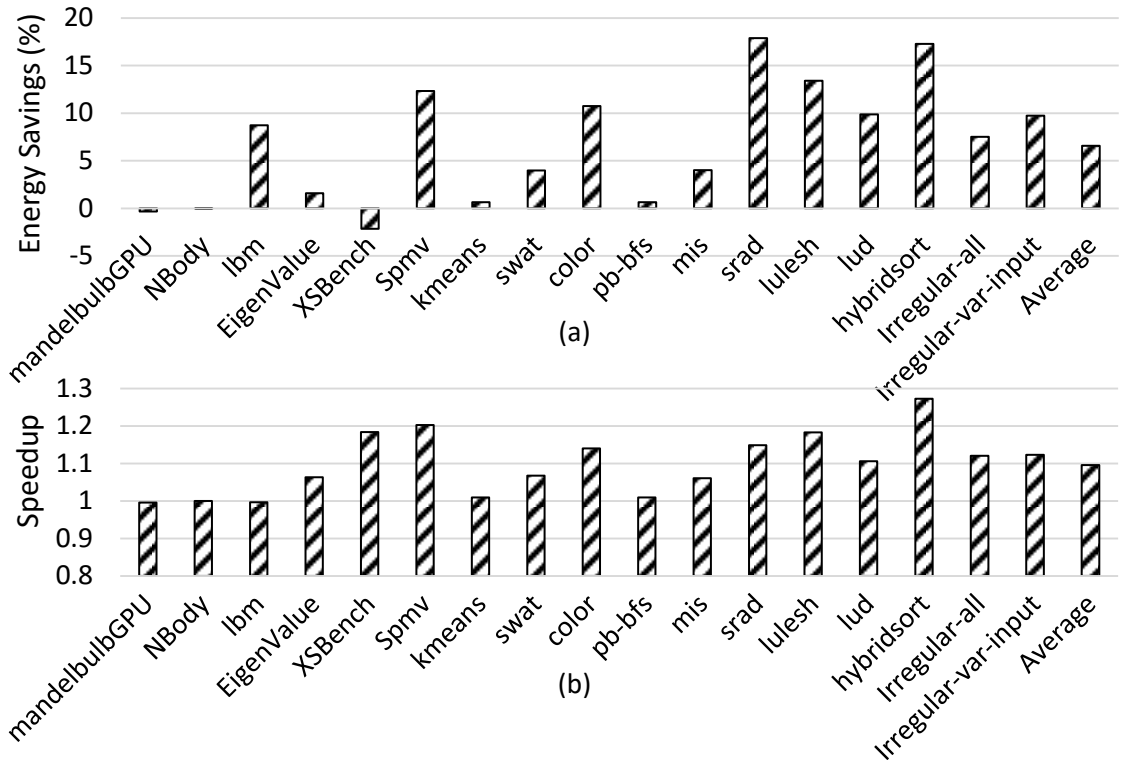


Figure 6.8: MPC (a) energy savings and (b) speedup over PPK.

Figure 6.8 shows the results of MPC with respect to PPK, which include the respective optimization overheads. Unlike the PPK approach described in Section 6.2.5 with perfect performance and power prediction, for a fair comparison, this version uses Random Forest for power and performance prediction as with MPC. Among the regular benchmarks, PPK works well for *mandelbulbGPU* and *NBody* because the same kernel is iterated and the kernels are input independent. For these benchmarks, MPC does not show an advantage.

MPC significantly outperforms PPK for the irregular benchmarks, where PPK often mispredicts the next kernel behavior, achieving 12% greater performance than PPK while simultaneously reducing energy by 7.5% for the 12 irregular benchmarks. For the irregular benchmarks, PPK suffers a 8–26% performance loss compared to AMD Turbo Core (Figure 6.7). This is due to next

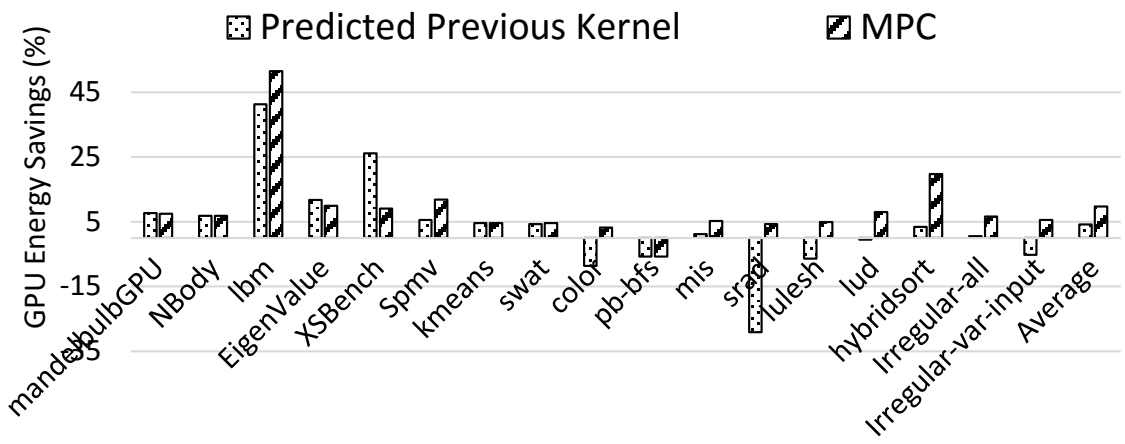


Figure 6.9: GPU energy savings over AMD Turbo Core.

kernel misprediction in conjunction with its inability to proactively change its decisions based on future kernel behavior. In contrast, MPC foresees the ability to catch up on the lost performance due to mispredictions in future kernels. For example, for *srad*, MPC outperforms PPK by 15%. MPC performs particularly well for the irregular benchmarks with kernels with varying input, outperforming PPK by 12.3% while reducing energy by 9.7%. For *XSBench*, MPC consumes more energy than PPK by choosing higher power configurations to reduce the performance loss. Overall, MPC outperforms PPK by 9.6% while reducing energy by 6.6%.

The CPU's contribution to the overall MPC energy savings over Turbo Core is 75%, while the GPU contributes 25%. This is because MPC intelligently lowers the CPU state as it does not improve the kernel execution time, whereas Turbo Core keeps the CPU at a higher DVFS state as long as the system is operating within its TDP limit. For this reason, we also show the GPU energy savings of MPC over Turbo Core in Figure 6.9. These energy savings also includes the static energy overhead of the GPU during MPC optimization.

The highest savings (51%) is achieved for *lbm* because its kernels exhibit peak

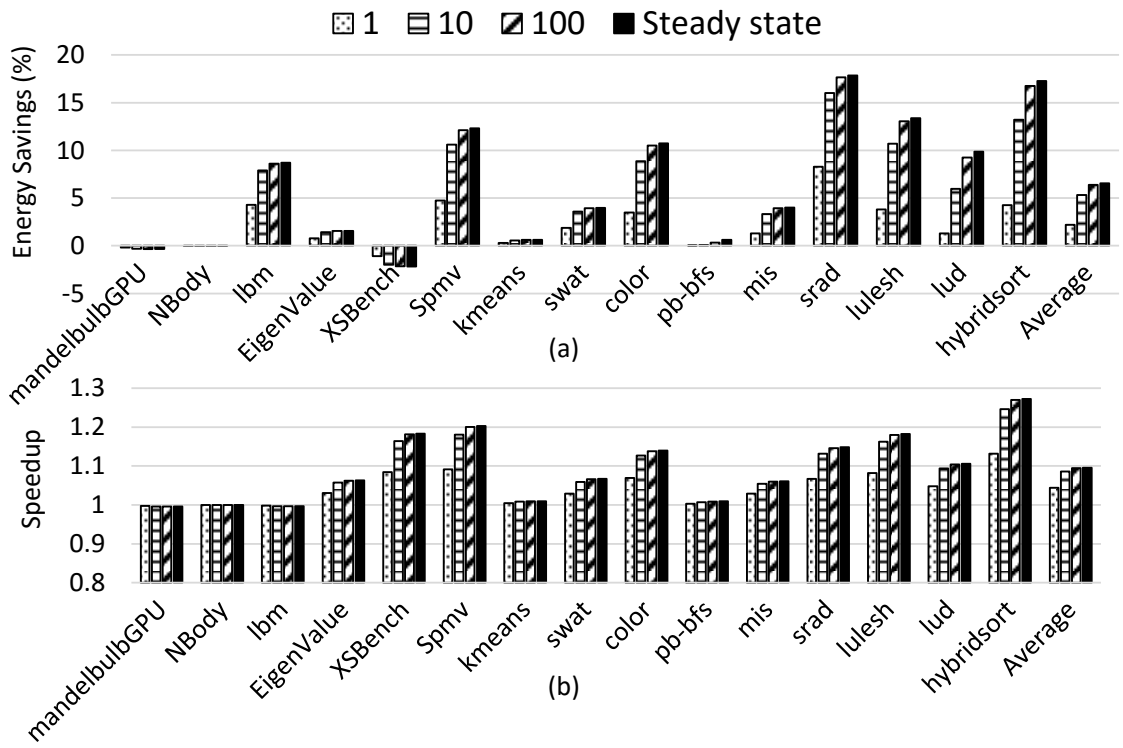


Figure 6.10: MPC (a) energy savings, and (b) speedup relative to PPK when the benchmarks are re-executed the specified number of times after the initial execution.

behavior. For other benchmarks, the savings is not as large, but still significant (3-20%), which leads to an overall energy savings of 10%. For *EigenValue* and *XSBench*, PPK shows higher GPU energy savings than its chip-wide savings. This is because PPK lowers the CPU and GPU power states while significantly increasing the execution time, thereby resulting in higher CPU energy. Compared with PPK, MPC achieves an average GPU energy savings of 5.1% while simultaneously improving performance by 9.6%.

6.6.2 Amortization of Initial Losses

Our approach benefits from repeated application execution to achieve gains. The initial losses of running PPK for the first execution can be amortized over these repeated executions. Figure 6.10 shows the energy savings and performance loss of MPC compared to PPK when the benchmarks are re-executed the specified number of times after the initial execution. The energy savings and performance loss includes the associated overheads. The steady state value is the ideal case with no initial losses during the profiling. Non-negligible gains are observed with just a single repeated execution, and most of the full gains are observed after only ten re-executions, indicating that MPC can significantly improve the energy efficiency of real-world workloads that repeatedly re-execute.

6.6.3 Comparison with Theoretical Limit

In this section, we explore how closely our polynomial-time heuristic MPC approach matches the theoretically achievable savings by comparing against the exponential-time Theoretically Optimal (TO) scheme. We assume perfect prediction, no MPC overhead, exhaustive search of all hardware configuration for each kernel, and a horizon length of all kernels. Figure 6.11 shows the results.

As expected, MPC performs similarly to TO for regular benchmarks. In general, MPC benefits from looking into the future behavior of all the kernels, and thus achieves near-optimal energy savings and performance gains. In particular, *pb-bfs*, *mis* and *lud* show lower energy savings than TO, while *EigenValue*, *mis* and *Spmv* suffer a slight performance loss. This is because the effectiveness of MPC is highly sensitive to its search order, which is derived based on the

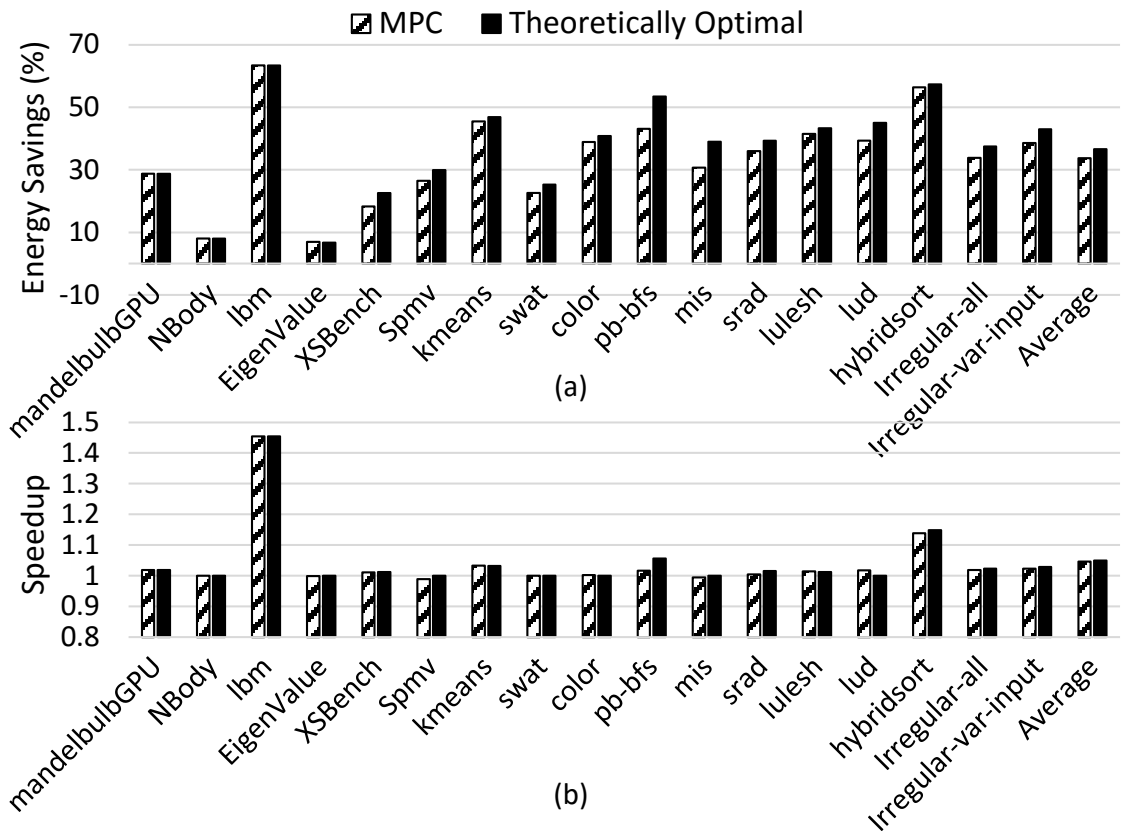


Figure 6.11: Comparison with Theoretical Limit. (a) Energy savings and (b) speedup over AMD Turbo Core.

sub-optimal PPK-based profiling. Overall, MPC achieves 92% of the maximum theoretical energy savings and 93% of the potential performance gain.

6.6.4 Ramification of Prediction Inaccuracy

The Mean Absolute Percentage Errors of our Random Forest prediction model over the 15 benchmarks are 25% and 12% for performance and power respectively. The high performance error is due to diverse performance scaling trends and the presence of outliers with unexpected performance behavior. In this section, we examine the potential loss in energy savings by our RF-based MPC compared to a MPC using a perfect prediction model. We consider a horizon

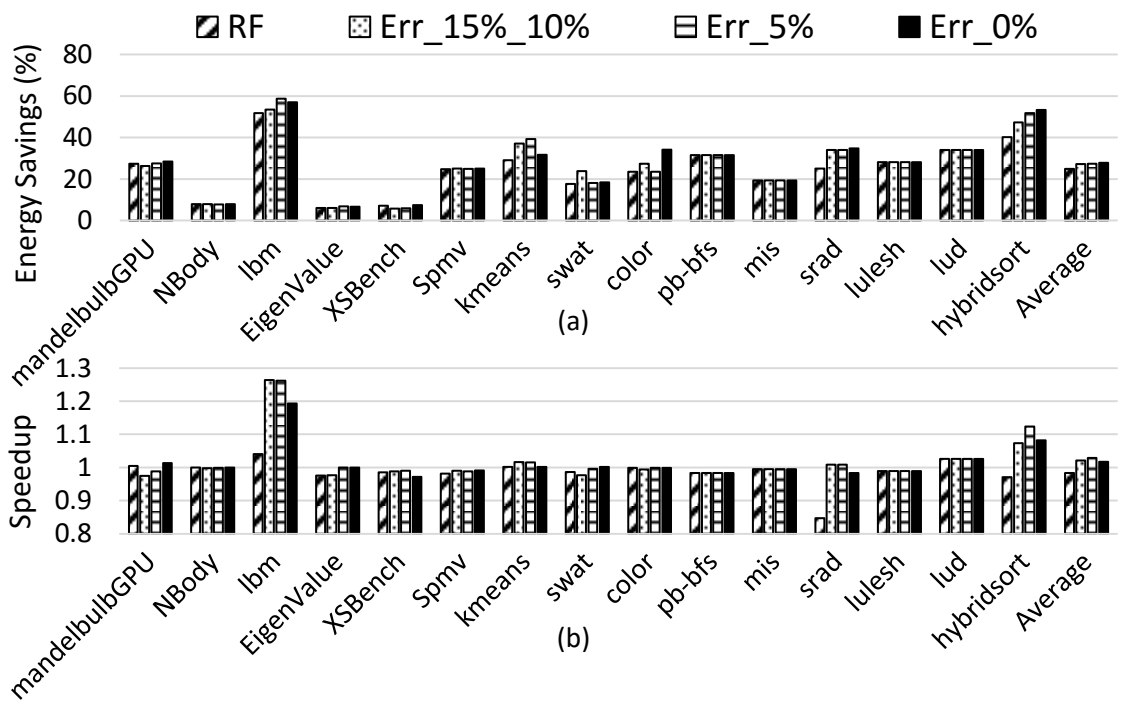


Figure 6.12: Ramification of prediction inaccuracy on energy-performance tradeoff.

length equal to the number of kernels and exclude the MPC overhead.

Figure 6.12 compares our Random Forest based MPC implementation (RF) with MPC implementations based on the accuracy of recently published prediction models. Err_15%_10% assumes prediction inaccuracies of 15% and 10% for performance and power respectively, as reported by Wu et al. [170]. Similarly, Err_5% considers prediction inaccuracy of 5%, as reported for Paul et al. [128]. A prediction model with no errors is represented by Err_0%. To implement these prediction models, we assume a half random normal distribution [31], with its absolute mean equal to the corresponding average error.

From Figure 6.12, RF behaves similar to Err_15%_10%. RF is better for *mandelbulbGPU* and *XSBench*, while Err_15%_10% is better for *kmeans*, *swat* and *srad*. On average, the energy savings of other models range from 27-28%, while RF's

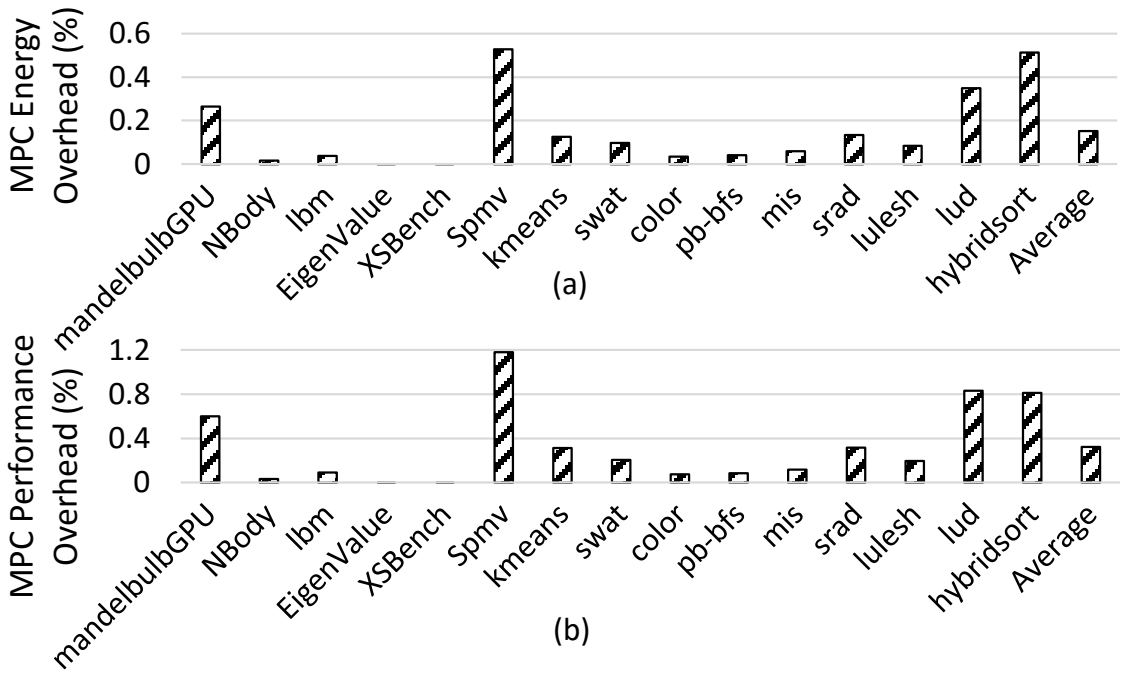


Figure 6.13: MPC (a) energy and (b) performance overheads with respect to Turbo Core.

savings is 25%. Similarly, other prediction models improve performance by 1.7-3%, while RF decreases performance by 1.7%. The reason for this small difference in optimality is because MPC uses the prediction models 65× less than an exhaustive search due to the combined effect of greedy hill-climbing and search order heuristic, and thus remains largely unaffected by model imperfections. It also takes the runtime performance as feedback and thus further rectifies the impact of these mispredictions by dynamically updating the performance headroom. The result is comparable energy savings with minor differences in performance.

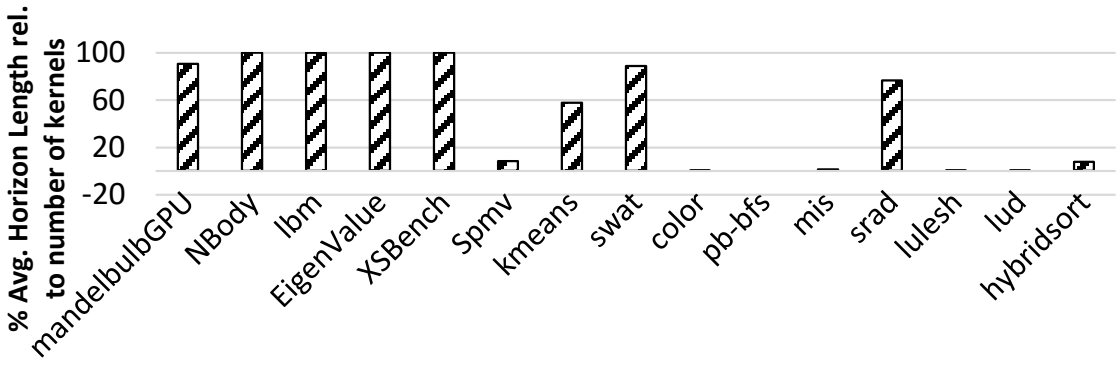


Figure 6.14: Average MPC horizon as a percentage of the total number of kernels.

6.6.5 MPC Overheads and Horizon Length

Figure 6.13 shows the MPC energy and performance overheads with respect to Turbo Core when adapting the horizon length for an α (performance slowdown factor) of 0.05 (5%). The average energy overhead is 0.15% (maximum of 0.53% for *Spmv*) with a performance overhead of 0.3% (maximum of 1.2% for *Spmv*). The overheads consider a worst case situation when kernels appear back-to-back with no CPU phases in between, or there are no available CPUs to run the algorithm during CPU phases. In reality, GPGPU application kernels may be separated by CPU phases with an available CPU, which can hide the MPC overheads. As a result, the actual overheads will be lower, permitting longer horizon lengths to improve performance.

Figure 6.14 shows the average MPC horizon length as a percentage of N, the total number of kernels in an application. Benchmarks *NBody*, *lbm*, *EigenValue* and *XSBench* have long kernel lengths, which permits MPC to explore the full horizon. For *MandelbulbGPU*, *kmeans* and *swat*, the horizon length generator initially selects shorter kernels before determining that there is enough performance margin to use the full horizon. The full horizon is initially selected for

srad, but lowered when encountering a performance loss due to misprediction. For the remaining benchmarks, the horizon length generator shrinks the horizon length significantly to limit the overheads since they have shorter kernel lengths.

We compare our adaptive horizon MPC scheme with one that uses full horizon. When ignoring overheads, the full-horizon MPC approach reduces energy by only 2.6% compared to our adaptive scheme, with similar performance impact. When the MPC overheads are included, the full-horizon scheme achieves a 15.4% energy savings over Turbo Core, with a performance loss of 12.8%, compared to 24.8% energy savings with a 1.8% performance loss for the adaptive scheme.

6.7 Conclusion

This chapter presents a dynamic power management scheme for GPGPU applications using Model Predictive Control (MPC). MPC anticipates future kernel behavior and makes proactive decisions to maximize energy efficiency with minimum impact on performance. We devise a variant of MPC that uses greedy and heuristic approximations, and adaptively tunes the horizon length to permit a low overhead practical runtime implementation. Our scheme achieves significant energy savings with negligible performance loss compared to the AMD Turbo Core power manager, and both energy savings and performance improvement over current history-based approaches.

CHAPTER 7

DYNAMIC POWER MANAGEMENT OF HETEROGENEOUS ARCHITECTURES

7.1 Introduction

A recent trend in high performance computing is to offload computation to an accelerator such as a GPU. For such applications, the CPU queues up work for the GPU and busy-waits, while the GPU performs the computation. For example, the microbenchmarks of SHOC test the compute or memory bandwidth of a GPU [42], and benchmarks suites such as AMD APP SDK [2], Rodinia [35] or Parboil [149] comprise applications such as *NBody* or *K-means* ported to the GPU.

Recently, GPU vendors such as AMD, NVIDIA and Qualcomm have introduced features such as unified memory [94], system-scope atomic operations in the GPU devices [5], and asynchronous kernel launches and memory copy [136], that permit programmers to improve application performance by using the CPU and the GPU concurrently. Programming models, such as Heterogeneous System Architecture [12] introduced by the HSA Foundation [9] allow programmers to more easily deploy both the CPU and the GPU, and have shown significant performance improvements [124, 154].

However, traditional power management schemes are not well tuned for such applications. State-of-the-practice power management techniques, such as AMD Turbo Core [49, 120] and Intel Turbo Boost [139, 137, 138] monitor chip resource usage and on-chip temperature to shift power to operate within the

TDP limit. Such schemes often waste energy by keeping the power states high if the chip is operating within the limit or degrade performance by prematurely throttling the power states.

State-of-the-art multi-programming schemes isolate the applications contending for shared resources before performing DVFS [104, 112, 113]. Since one hardware setting is not optimum for applications with contrasting characteristics operating simultaneously, determining an appropriate setting that works jointly for all of them is crucial. While the performance of the contending applications slows down relative to their isolated execution [113, 153], we observe separating them does not necessarily improve energy efficiency.

In this chapter, we extend our MPC-based power management scheme proposed in Chapter 6, for heterogeneous applications that simultaneously use the CPU and the GPU. Our proposed scheme attempts to improve the energy efficiency of heterogeneous applications without impacting its performance relative to a given target. Our MPC power manager uses a performance and power predictor coupled with profiled baseline performance data to dynamically track ongoing performance relative to the target, and seeks opportunities from the future to impact its current decisions. Our scheme determines the hardware settings of future sub-phases of the application and uses the performance credit to expand or shrink the upcoming sub-phase in order to avoid performance loss. We examine three MPC search heuristics that differ in their treatment of different sub-phases within the application. Our MPC scheme also adaptively varies its horizon length to keep its overheads low.

We compare our MPC policies to two baselines: (a) a state-of-the-art policy that separates the memory-bound phases, and (b) a new rule-based policy that

runs the memory-bound phases together. With respect to the first baseline, we achieve 24% energy savings while improving performance by 13.3%. With reference to the second baseline, our MPC scheme reduces energy by 17.2% while improving performance by 9.4%.

7.2 Background

In this section, we examine the characteristics of real-world heterogeneous workloads, and use this information to generate synthetic heterogeneous workloads. We characterize their nature of computation, demonstrate the performance impact of simultaneously utilizing the on-chip compute resources, and illustrate the limitations of state-of-the-art power management policies that aim to improve energy efficiency for such heterogeneous applications.

7.2.1 Real-world Heterogeneous Applications

Real-world heterogeneous applications, such as *gromacs* that simulates molecular dynamics [123], and *linpack* that simulates numerical linear algebra use the CPU and the GPU when implemented on the AMD A10-7850K APU. Figures 7.1 and 7.2 show the Application Programming Interface (API) traces of these two applications collected from AMD CodeXL [4]. Each row corresponds to the OpenCL function called by either the CPU or the GPU along with the application execution time. The host threads represent the CPU executing the application. The colored rectangles correspond to the OpenCL function and their execution time, while the in-between empty spaces represent that the CPU is

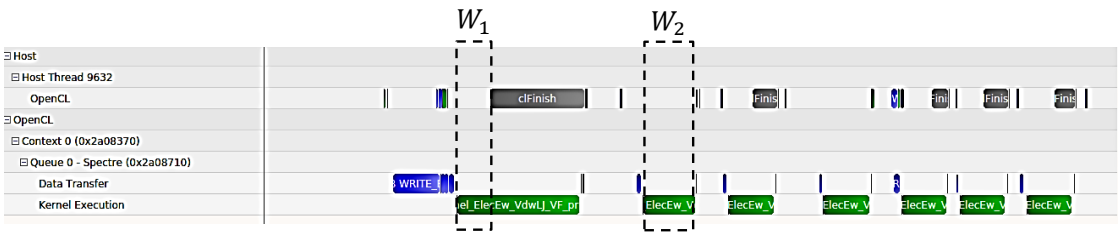


Figure 7.1: API trace of application *gromacs*.

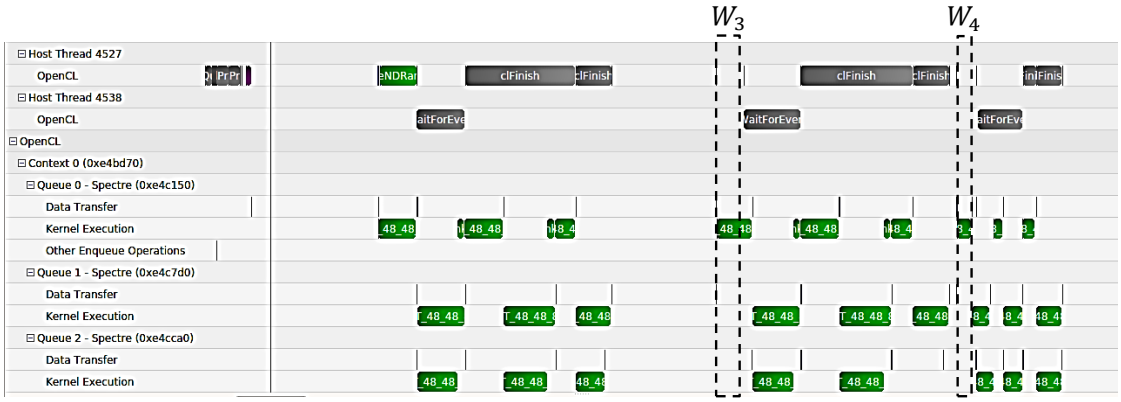


Figure 7.2: API trace of *linpack*.

busy executing the application. The function *clFinish()* blocks the CPU execution until all previously queued OpenCL commands are issued to the GPU and have completed [3]. This represents that the CPU is busy waiting until the GPU has finished executing the kernel. The *Data Transfer* row shows the time period of transferring the data between the CPU and the GPU memory, while the *Kernel Execution* row represent the time GPU is busy executing the kernel. The region of interest are marked with a dashed box.

As shown from the API trace of *gromacs* in Figure 7.1, at window W_1, the OpenCL call *clFinish()* asynchronously starts while the corresponding kernel *nbnxn_kernel_ElecEw_VdwLJ_VF_prune_opencl* is in execution. Until the time *clFinish()* starts, the CPU is doing some other execution in parallel. Similarly, the next iteration of this kernel at window W_2 runs while the CPU is executing in parallel. The corresponding *clFinish()* call starts after the kernel execution is over. On the

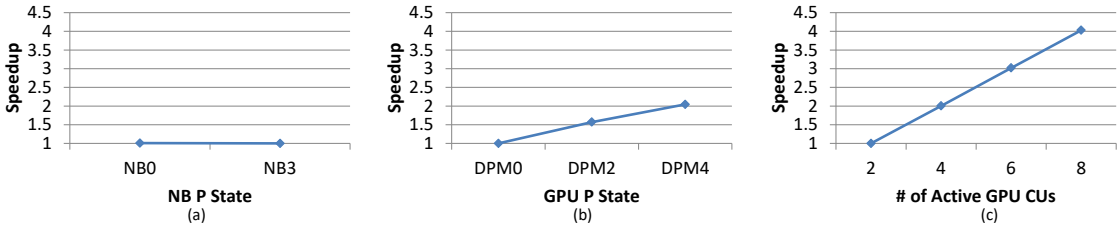


Figure 7.3: Performance scaling of kernel *nbnxn_kernel_ElecEw_VdwLJ_VF_prune_opencl* of application *gromacs* across different GPU hardware configurations. The speedup is with respect to the slowest performance.

other hand, *linpack* runs concurrent kernels as shown in Figure 7.2. Additionally, we see simultaneous execution of GPU kernel *dgemm_NT_48_48_8_8x8_6x6* and the CPU thread at windows W_3 and W_4.

Furthermore, from Figure 7.3, we observe that the kernel of *gromacs* is compute-bound because its performance scales with GPU DVFS states and CUs, while it remains invariant to changes in NB DVFS states. However, from Figure 7.4, the kernel of *linpack* is both compute- and memory-bound, and also shows a performance peak behavior when the CU count equals six. Moreover, from Figure 7.6, the corresponding CPU phases are more memory-bound in nature as their memory intensities (defined later) are more than 15%[1]. Motivated by the simultaneous execution of phases with diverse performance characteristics in heterogeneous applications, and that of the mixture of applications scheduled within a node of a datacenter composed of CPUs and GPUs [25, 45, 135, 141, 146, 148, 166], we generate synthetic workloads using all possible combinations of compute- and memory-bound overlappings to reflect the behavior of real-world heterogeneous applications.

[1]In our experiments, compute-bound *gamess* and *hmmer* show less than 1% *mem_intensity*, while memory-bound *C_lbm* and *mcf* show a *mem_intensity* of 30%.

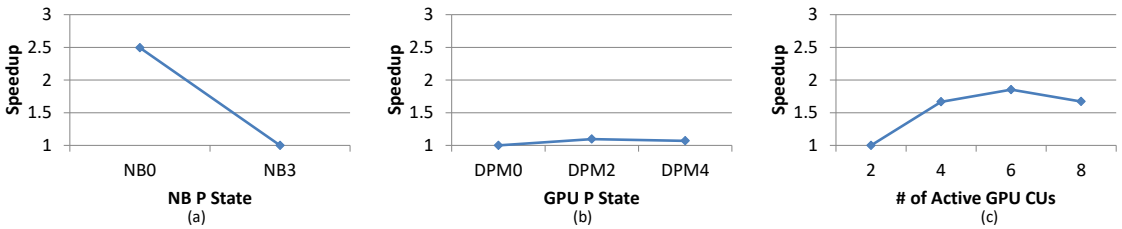


Figure 7.4: Performance scaling of kernel *dgemm_NT_48_48_8_8x8_6x6* of application *linpack* across different GPU hardware configurations. The speedup is with respect to the slowest performance.

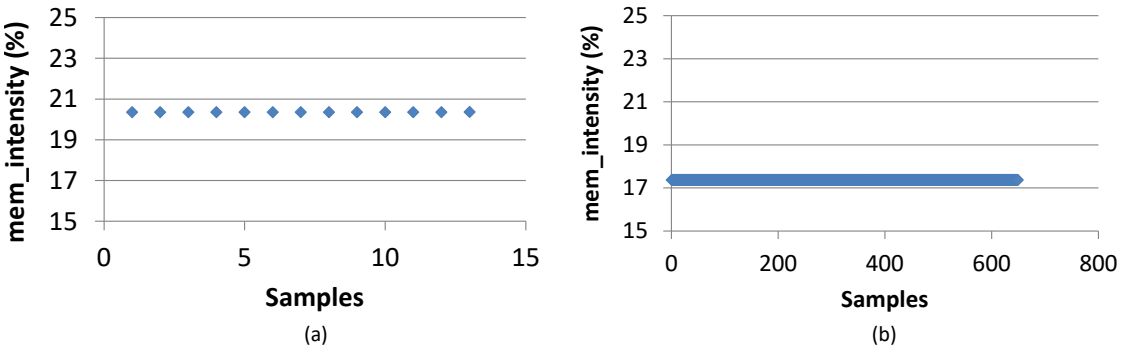


Figure 7.5: Memory intensity of the simultaneous CPU portion while the GPU kernel is executing for (a) *gromacs* and (b) *linpack* at the maximum APU hardware configuration.

7.2.2 SPEC-GPGPU based Heterogeneous Application Phases

To represent real heterogenous applications, we create synthetic phases that are composed of a SPEC CPU 2006 benchmark [69] executed with *train* mode and a GPGPU benchmark running in parallel. Table 7.1 shows the list of SPEC CPU 2006 and GPGPU benchmarks considered in this chapter, along with their nature of computation [107, 131]. We iterate the GPGPU benchmark multiple times while the SPEC benchmark is running.

Figure 7.6 shows the relative performance of the SPEC CPU and the GPU kernel when they are executed together with respect to their isolated execution. From Figure 7.6, we note that overlapping compute-bound SPEC with compute-bound GPGPU benchmarks, such as *gamess* or *hmmer* with *NBody*, do

Table 7.1: SPEC and GPGPU benchmarks used to create heterogeneous workloads. *C_lbm* represents *lbm* from the SPEC CPU 2006 benchmark suite [69]. *G_lbm* represents *lbm* from the Parboil benchmark suite [149]. NBody is from AMD APP SDK [2] and *XSBench* is an Exascale application [13].

	Benchmarks	**Type**
SPEC	hmmer	Compute-bound, Integer
	gamess	Compute-bound, Floating Point
	C_lbm	Memory-bound, Integer
	mcf	Memory-bound, Floating-Point
GPGPU	NBody	Compute-bound
	G_lbm	Memory-bound
	XSBench	Memory-bound

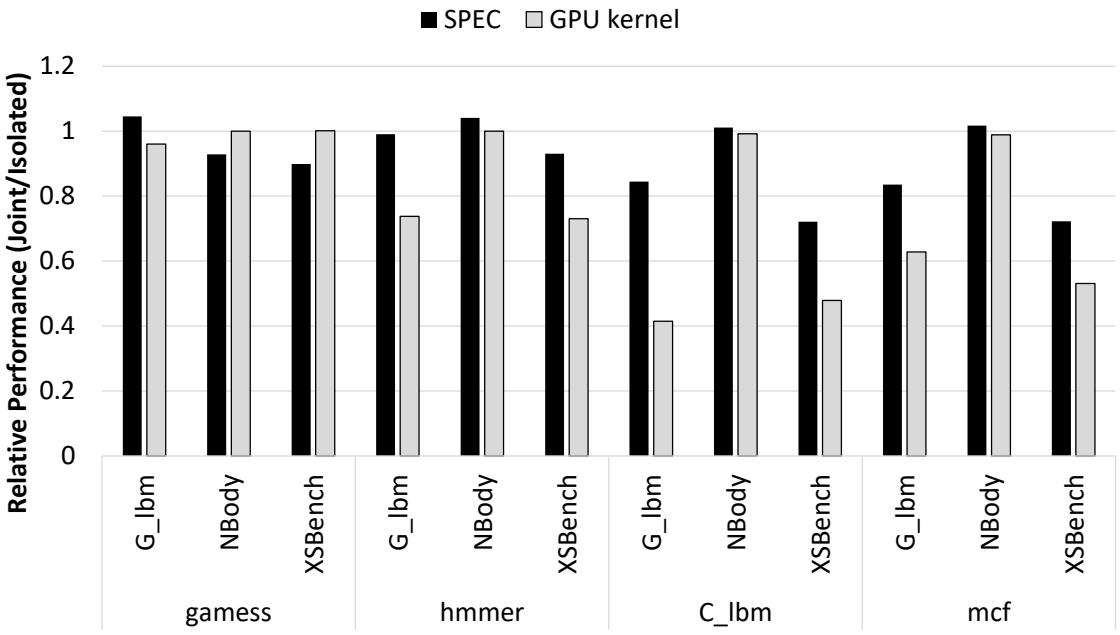


Figure 7.6: Relative performance of the SPEC CPU and the GPU kernel when executed together with respect to its isolated execution at maximum hardware configuration.

not cause significant slowdown. However, overlapping a memory-bound SPEC with a memory-bound GPGPU benchmark causes significant slowdown. For instance, overlapping *mcf* with *XSBench* causes 28% slowdown for *mcf* and *47%* performance loss for *XSBench*. This is due to conflicts for the memory channel. Motivated by this finding, state-of-the-art power management approaches pro-

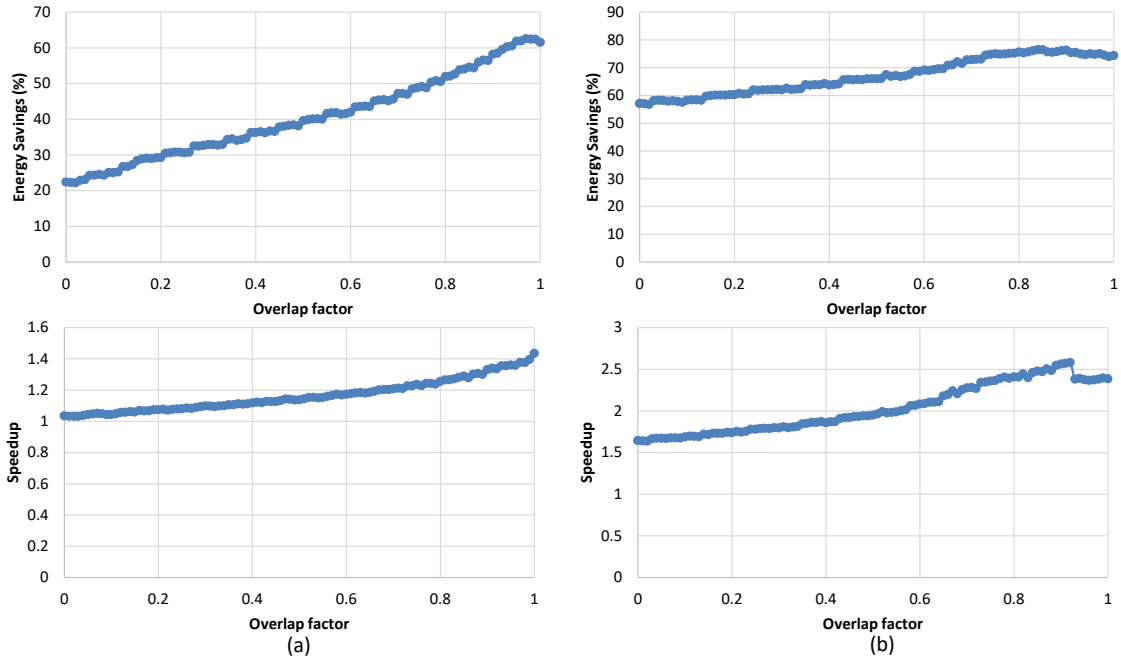


Figure 7.7: Energy and performance behavior of *MPC_1* for (a) *mcf* and *G_lbm*, and (b) *mcf* and *XSBench*. The overlap factor indicates the fraction of overlap. A one indicates full overlap, while a zero indicates complete separation.

pose to isolate the conflicting phases and perform DVFS for these individual portions, while joint DVFS is done for non-conflicting phases [112, 113].

Based on the state-of-the-art baseline that separates the memory phases, we analyze the impact of the overlap of memory-bound phases on the energy savings and performance using the MPC policy with perfect prediction and a horizon length of one. Our baseline policy keeps the power states of compute resources to the lowest and memory to the highest, and therefore uses the hardware configuration [P7, NB0, DPM0, CU2].

From Figure 7.7, we observe that the maximum energy savings occurs with close to full overlap for *mcf* and *G_lbm*, while the maximum energy savings for *mcf* and *XSBench* occurs when they are 90% overlapped. This is because the kernel of *G_lbm* constitutes around 30% of the total time, while that of *XS-*

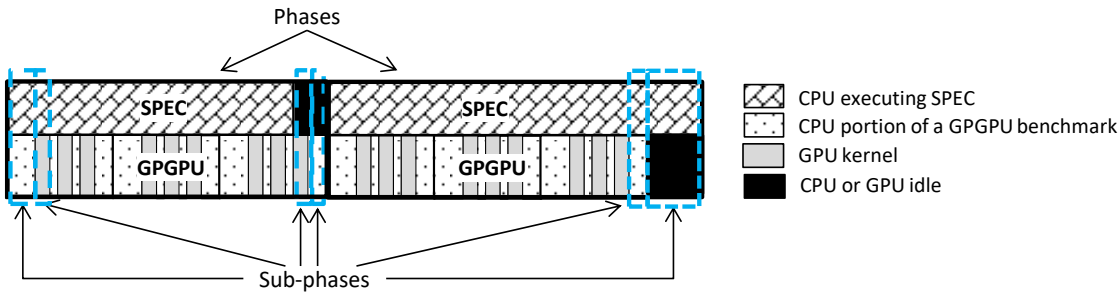


Figure 7.8: Schematic of a heterogeneous workload that is composed of multiple *phases*. Each phase constitutes the overlap of a SPEC benchmark and multiple iterations of a GPGPU benchmark. Within a phase, there exists multiple *sub-phases*. A sub-phase can either be an overlap of SPEC benchmark with the GPU kernel or the CPU portion of GPGPU benchmark. A sub-phase can also be the GPU kernels or the CPU portion of the GPGPU benchmark running in isolation, or the SPEC benchmark running solely.

Bench constitutes around 65%. As a result, the former has a smaller portion that leads to memory interference with the CPU thread, while the interference for the latter is much more significant. Inspired by these findings, we compare our proposed power management policy to two baseline policies: (a) a state-of-the-art policy that separates the memory-bound phases, and (b) a new rule-based policy that keeps the memory-bound phases unseparated. Our new baseline consumes slightly more energy with a negligible performance impact over the scheme that optimally overlaps the memory-bound phases.

7.3 Problem Statement

In this section, we define the common terms used in this chapter and present the mathematical formulation of our problem.

7.3.1 Definitions

As introduced in Section 7.2, a *phase* constitutes overlapped execution of a SPEC benchmark with repeated executions of a GPGPU benchmark. Our heterogeneous workloads are composed of multiple such *phases*, shown in Figure 7.8. Our MPC-based power manager makes decisions at a *sub-phase* level, which we define as the overlapped execution of either a GPU kernel or the CPU portion of the GPGPU benchmark with the SPEC benchmark. A *sub-phase* can also include GPU kernels or the CPU phase of the GPGPU benchmark with no SPEC benchmark running in parallel, or a SPEC benchmark with no overlapping GPGPU benchmark. We assume that the synchronization points occur at the phase boundary. In other words, we assume that there is no dependency between the CPU and the GPU within a phase, as demonstrated by real-world applications, such as Blackscholes, Gene Alignment, Evolutionary Programming and Image Background Extraction, from the Hetero-mark benchmark suite [154]. A phase must execute entirely before other phases are launched.

7.3.2 Formulation

The overall objective is to minimize the energy of the heterogeneous application composed of multiple phases such that the execution time does not exceed that of a baseline and the application still does the same amount of work. In order to represent the amount of work, we use number of SPEC instructions executed by the CPU and the number of GPGPU calls made. We achieve energy efficiency by suitably selecting the DVFS states of the CPU, Northbridge and GPU, and the number of active GPU compute units, at sub-phase granularity. Equation 7.1

presents the formulation.

$$
\begin{aligned}
&\min_{\vec{s}} \sum_{i=1}^{P} \sum_{j=1}^{P_i} E_{i,j}(s_{i,j}) \\
&\text{such that} \\
&\sum_{i=1}^{P} \sum_{j=1}^{P_i} T_{i,j}(s_{i,j}) \leq T_B \\
&\sum_{j=1}^{P_i} I_{i,j}(s_{i,j}) = I_{B_i}, \quad \forall 1 \leq i \leq P \\
&G_i = G_{B_i}, \quad \forall 1 \leq i \leq P \\
&\text{where } s_i \in S \text{ and } S = \overrightarrow{cpu} \times \overrightarrow{nb} \times \overrightarrow{gpu} \times \overrightarrow{cu}
\end{aligned}
\tag{7.1}
$$

In Equation 7.1, an application contains P phases. Each phase P contains P_i sub-phases. $E_{i,j}$ represents the energy of the j^{th} sub-phase of the i^{th} phase, while $s_{i,j}$ indicates its hardware configuration. Mathematically, $s_{i,j}$ is an element of the vector $\vec{s_i}$, which belongs to the set S of possible hardware configurations. S is a Cartesian product of $\overrightarrow{cpu}$, $\overrightarrow{nb}$, $\overrightarrow{gpu}$ and $\overrightarrow{cu}$, which represent the CPU, NB, and GPU DVFS states, and the number of ways the GPU CUs can be activated, respectively.

For the j^{th} sub-phase of the i^{th} phase, $T_{i,j}$ represents the execution time and $I_{i,j}$ represents the numbers of SPEC instructions executed by the CPU. G_i indicates the number of GPGPU calls made within the i^{th} phase. T_B indicates the execution time of the entire application when the hardware is configured according to a baseline policy. I_{B_i} and G_{B_i} are the number of SPEC instructions executed by the CPU and the number of GPGPU calls of the i^{th} phase when it is executed under the baseline power management policy. The details of our baseline policies is provided in Section 7.5.

If N_i represents the number of kernels in one GPGPU call of the i^{th} phase, the number of sub-phases within this one GPGPU benchmark is $2N_i + 1$, with $N_i + 1$ CPU sub-phases. For G_i GPGPU calls, the total number of sub-phases is $G_i \times (2N_i + 1)$. For M possible hardware configurations, the complexity of finding energy-optimal configurations for just one phase is $\mathrm{O}\left(M^{\cdot^{\cdot^{\cdot^{M}}}}{}^{G_i \times (2N_i+1) \text{ times}}\right)$. This is because the exact amount of overlap between the SPEC and GPGPU benchmarks in a sub-phase depends on previous overlaps. The complexity grows further when we consider applications with multiple phases. Exponential complexity of this nature renders finding a theoretically optimal solution challenging and infeasible to implement. Motivated by this finding, we relax Equation 7.1 to Equation 7.2, which is more tractable and is of polynomial-time complexity.

$$
\begin{aligned}
&\min_{s_{i,k} \in S} E_{i,k}(s_{i,k}) \\
&\text{such that} \\
&\sum_{k=1}^{j} I_{i,k}(s_{i,k}) \geq \sum_{k=1}^{j} I_{B_k} \\
&\sum_{k=1}^{j} T_{i,k}(s_{i,k}) \leq \sum_{k=1}^{j} T_{B_k} \\
&\forall 1 \leq k \leq j,\ \forall 1 \leq j \leq P_i,\ \forall 1 \leq i \leq P, \text{ and } \forall s_{i,k} \in S
\end{aligned}
\tag{7.2}
$$

In Equation 7.2, for the j^{th} sub-phase, the optimization algorithm chooses the hardware configuration that minimizes the energy of that sub-phase while ensuring that its total SPEC instruction count thus far (including that of previous sub-phases) at least exceeds that of the SPEC instructions processed by the baseline when the same sub-phase was executed. The cumulative execution time until the j^{th} sub-phase should also be less than the cumulative baseline execution time of the j^{th} sub-phase. We also ensure that the number of GPGPU

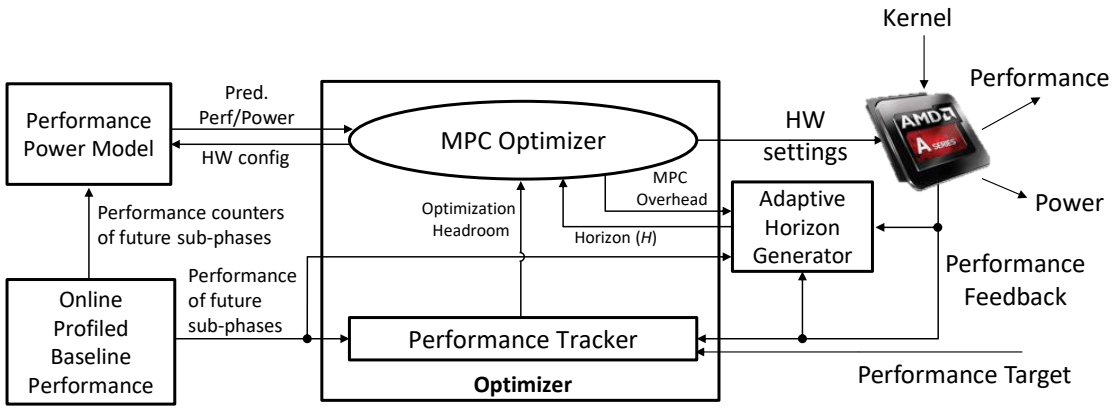


Figure 7.9: System architecture of MPC-based power manager.

calls at the end of the phase must incrementally match that of the baseline (not shown in the above equation). The complexity of this approach for an i^{th} phase is $O(M \times G_i \times N_i)$.

7.4 Power Management Architecture

In this section, we describe the architectural details of our MPC-based power manager. Figure 7.9 shows the architecture. The key function of our power manager is to determine a hardware setting before a sub-phase is invoked that reduces the system-level energy consumption of a heterogeneous application, while not sacrificing performance. The hardware setting is determined by the optimizer. The optimizer takes the performance target as input, and uses the performance and power model, profiled baseline performance data, and appropriate horizon from Adaptive Horizon Generator to determine the hardware setting. In the following sections, we describe each of these architectural components in detail.

7.4.1 MPC Optimizer

The *optimizer* works on the principle of MPC as described in Section 6.4. For the upcoming j^{th} sub-phase, the *optimizer* looks into a window of H_j future sub-phases, finds the hardware configuration for each of these sub-phases such that the accumulated execution times including that of the past execution times is less than the baseline execution time of those sub-phases, and aims to execute at least the same number of instructions as executed by the baseline policy. Based on accumulated performance and instruction credit, it determines the appropriate setting of the j^{th} sub-phase. The objective function of the *optimizer* is expressed by Equation 7.3.

$$\min_{\vec{s} \in S} \sum_{k=1}^{j+H_j-1} E_{i,k}(s_{i,k})$$

such that

$$\begin{aligned} \sum_{k=1}^{j+H_j-1} I_{i,k} &\geq \sum_{k=1}^{j+H_j-1} I_{B_k} \\ \sum_{k=1}^{j+H_j-1} T_{i,k} &\leq \sum_{k=1}^{j+H_j-1} T_{B_k} \end{aligned} \tag{7.3}$$

$$\forall 1 \leq k \leq j,\ \forall 1 \leq j \leq P_i,\ \forall 1 \leq i \leq P, \text{ and } \forall s_{i,k} \in S$$

MPC Search Order Heuristic As described in Section 7.3, Equation 7.1 is exponential in nature, which we convert to Equation 7.2 to make it tractable. However, looking into the future involves backtracking and would again make the complexity exponential. For a tractable MPC implementation, we propose a strict order of optimizing the future sub-phases such that a sub-phase is never reoptimized. In this chapter, we propose three search order heuristics. All of our search heuristics optimize the slowest sub-phase first. However, they differ

in their approach based on the duration of the phases involved.

- **Flat:** The *flat* heuristic orders sub-phase from longest to shortest execution time without considering phase boundaries. To understand this better, we take an example benchmark shown in Figure 7.10 with two phases A and B. The phase A contains sub-phases A_1 and A_2, while phase B contains sub-phases B_1, B_2 and B_3. The *flat* search heuristic optimizes the sub-phases in the decreasing order of baseline execution times, which is (B_1, A_2, B_3, B_2, A_1).
- **Coarse:** The *coarse* heuristic orders the sub-phases within the slowest phase first. Within the slowest phase, the sub-phases are optimized sequentially. In the above example, phase B is slower than A, therefore sub-phases of B will be optimized sequentially before the sub-phases of A. As a result, the search order would be (B_1, B_2, B_3, A_1, A_2).
- **2-level:** The *2-level* heuristic prefers the slowest phase first. Within the slowest phase, the slowest sub-phases are optimized first. In the above example, phase B will be optimized before A. Within B, the order will be (B_1, B_3, B_2). Within phase A, the order will be (A_2, A_1). The overall order produced by *2-level* search heuristic will be (B_1, B_3, B_2, A_2, A_1).

The *flat* heuristic does not distinguish between the phases, which could result in reduced energy savings. Furthermore, the *flat* heuristic looks into the slower sub-phases from the entire application to determine the search order, which can produce longer horizons and therefore larger overheads. The *coarse* heuristic, however, prioritizes the slowest phase first, but disregards the heterogeneity among its sub-phases, which may again result in reduced savings. However, the *coarse* heuristic orders the sub-phases sequentially within

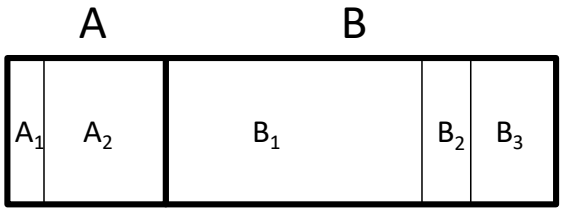


Figure 7.10: A benchmark example to illustrate the search order heuristic.

the slower phase, which results in a shorter distance between the current sub-phase and that of the future sub-phase. As a result, the *coarse* heuristic produces shorter horizons with smaller overheads. The *2-level* heuristic balances the limitations of the *flat* and *coarse* heuristics by considering both phase- and sub-phase level heterogeneity, and is likely to show greater savings with larger overheads than *coarse*, and larger savings and smaller overheads than that of the *flat* heuristic.

Once a search order is determined, the MPC operation works exactly as explained in Section 6.4.1.

Performance Tracker The *performance tracker* component within the optimizer block determines the headroom of the optimization of a sub-phase, which is used by the MPC optimizer to determine the setting of an upcoming sub-phase. The performance tracker determines the instruction and performance headroom based on the difference between the performance and instruction credit of the past and future sub-phases with that of the baseline execution time and instruction count, as shown in Equation 7.4. Here, $\mathbb{E}[I_{i,k}]$ and $\mathbb{E}[T_{i,k}]$ indicate the expected SPEC instruction count and expected execution time of the k^{th} future

sub-phase.

$$I_{j,k} \geq \sum_{k=1}^{j+H_j-1} I_{B_k} - \sum_{k=1}^{j-1} I_{i,k} - \sum_{k=j+1}^{j+H_j-1} \mathbb{E}[I_{i,k}]$$
$$T_{j,k} \leq \sum_{k=1}^{j+H_j-1} T_{B_k} - \sum_{k=1}^{j-1} T_{i,k} - \sum_{k=j+1}^{j+H_j-1} \mathbb{E}[T_{i,k}] \tag{7.4}$$

If MPC accelerates past and future sub-phases with respect to the baseline configuration, it could slow down the upcoming sub-phase to save energy. Similarly, if MPC is expected to run more instructions from the past and future sub-phases than that of the baseline, it may execute fewer instructions. On the contrary, if due to mispredictions, not much margin is left because of past decisions and from an anticipated performance impact of future sub-phases, MPC may accelerate the current sub-phase. In the event when there is no margin left, MPC selects a fail-safe configuration. The details of the fail-safe configuration are presented in Section 7.5.

7.4.2 Performance-Power Model

In this section, we describe the *performance and power model* that estimates the power, performance and instruction count of any sub-phase across any hardware configuration. The model takes the corresponding baseline data and scales it to a desired hardware configuration.

Performance Predictor

Our SPEC performance predictor is inspired by the PPEP model proposed by Su et al. [151]. Table 7.2 shows the performance counters we use in the model. We

Table 7.2: Performance counters used in predicting the performance and power of the CPU and the GPU.

	Performance Counters	Description
CPU	MAB_Wait_Cycles	Miss Address Buffer based Leading Loads Cycles, which is the cycles when the first non-speculative load misses the last-level cache and when the data is returned back.
	CPU_Clock_not_Halted	CPU clock cycles.
	Retired_Instr_User	Retired Instruction Count.
GPU	VALUBusy	Percentage of the GPU time vector ALU instructions are processed.
	VALUUtilization	Percentage of active vector ALU threads in a wave, indicates branch divergence.
	MemUnitBusy	Percentage of the total GPU time the memory fetch/read unit is active, including stalls and cache effects

use the *MAB_Wait_Cycles* and *CPU_Clock_not_Halted*, which indicate the leading loads cycles [150, 151] and the CPU clock cycles of a sub-phase, respectively, to compute the memory intensity, as shown in Equation 7.5. The *mem_intensity* metric represents the number of cycles the CPU is waiting for the data miss from the last-level cache to be resolved. As a result, it can be used to identify the percentage of CPU cycles that is sensitive to the memory frequency.

$$mem\_intensity = \frac{MAB\_Wait\_Cycles}{CPU\_Clock\_not\_Halted} \tag{7.5}$$

Using the *mem_intensity* factor, we estimate the cycles-per-instruction (CPI) of the CPU phase associated either with the SPEC or the GPGPU benchmark. Depending on the nature of overlap captured from the profiled baseline data, the baseline CPI of either the SPEC benchmark or that of the CPU phase of the GPGPU benchmark is used. If the baseline configuration of a sub-phase overlapped both the GPGPU benchmark and the SPEC benchmark or if the GPGPU benchmark was running in isolation, the baseline CPI of the CPU phase of the

GPGPU benchmark is used. However, if the sub-phase consist of the SPEC benchmark running in isolation, we use the baseline CPI of the SPEC benchmark.

Using the PPEP model, we project the CPI of a sub-phase to a new CPU DVFS state. We also include the NB DVFS state, as shown in Equation 7.6.

$$CPI\,(f_{CPU}, f_{NB}) = CPI_{baseline} \times \left[(1 - mem\_intensity) + mem\_intensity \times \frac{f_{CPU}}{f_{CPU,\,baseline}} \times \frac{f_{NB,\,baseline}}{f_{NB}}\right] \tag{7.6}$$

The CPU execution time of a sub-phase is estimated by using the predicted CPI from Equation 7.6 in Equation 7.7.

$$T_{CPU}\,(f_{CPU}, f_{NB}) = T_{baseline} \times \frac{CPI\,(f_{CPU}, f_{NB})}{CPI_{baseline}} \times \frac{f_{CPU,\,baseline}}{f_{CPU}} \tag{7.7}$$

Similarly, we use the GPU performance counters *VALUBusy* and *VALUUtilization* to represent the compute intensity, and *MEMUnitBusy* to indicate the memory intensity. Our prediction model is inspired from the *C-to-M Intensity* factor proposed in Harmonia [128]. We use these counters to scale the GPU kernel execution time to any NB and GPU hardware configuration, as shown in Equation 7.8.

$$T_{GPU}\,(f_{NB}, f_{GPU}, CU_{GPU}) = T_{baseline} \times \frac{CU_{GPU,\,baseline}}{CU_{GPU}}$$
$$\left[MEMUnitBusy \times \frac{f_{NB,\,baseline}}{f_{NB}} + VALUUnitBusy \times \frac{VALUUtilization}{100} \times \frac{f_{GPU,\,baseline}}{f_{GPU}}\right] \tag{7.8}$$

Recall that a subphase can either be the SPEC benchmark overlapping with the GPU kernel or the CPU phase of the GPGPU benchmark. Moreover, a sub-phase can also be a GPU kernel running in isolation or the SPEC benchmark running solely. To estimate the execution of a sub-phase, we monitor the corresponding baseline GPU performance counters. If no GPU performance counters are available (indicated by ϕ in Equation 7.9), it indicates that the sub-phase does not contain a GPU kernel and must be executing either the CPU phase of the GPGPU benchmark or the SPEC benchmark. As a result, the predicted time is the estimated CPU execution time. However, if baseline GPU counters are available, we use the predicted GPU kernel time as the predicted sub-phase time. Equation 7.9 shows this relationship.

$$T(f_{CPU}, f_{NB}, f_{GPU}, CU_{GPU}) = \begin{cases} T_{CPU}(f_{CPU}, f_{NB}) & \text{, if } GPUPerfCounters_{baseline} = \phi \\ T_{GPU}(f_{NB}, f_{GPU}, CU_{GPU}) & \text{, otherwise} \end{cases} \tag{7.9}$$

Instruction Count Predictor

The *mem_intensity* factor can also be used to predict the SPEC instruction count of a sub-phase for a new hardware configuration based on the baseline instruction count. We use the *Retired_Instr_User* performance counter to monitor the instructions processed for a sub-phase. Using the predicted CPI from Equation 7.6 and the estimated execution time of the sub-phase from Equation 7.9, Equation 7.10 is used to predict the SPEC instruction count.

$$I_{CPU}\left(f_{CPU}, f_{NB}, f_{GPU}, CU_{GPU}\right) = I_{CPU,baseline} \times \left[\frac{T(f_{CPU}, f_{NB}, f_{GPU}, CU_{GPU})}{T_{CPU,baseline}} \times \frac{CPI\left(f_{CPU}, f_{NB}\right)}{CPI_{baseline}} \times \frac{f_{CPU}}{f_{CPU,\ baseline}}\right] \tag{7.10}$$

Power Predictor

Recall from Section 6.2 that the AMD A10-7850K APU shares the same voltage rail for the Northbridge and the GPU. As a result, changing the DVFS state of the Northbridge impacts the voltage of the GPU, and vice versa. Equation 7.11 represents the relationship of the impacted voltage when the NB and GPU DVFS settings are changed.

$$V_{NB\_GPU}\left(V_{NB}, V_{GPU}\right) = \begin{cases} \max\left[\max\left(V_{NB}, V_{GPU}\right), V_{DPM4}\right] & \text{, if NB = NB0} \\ \max\left[\max\left(V_{NB}, V_{GPU}\right), V_{DPM3}\right] & \text{, if NB = NB1} \\ \max\left[\max\left(V_{NB}, V_{GPU}\right), V_{DPM2}\right] & \text{, if NB = NB2} \\ \max\left(V_{NB}, V_{GPU}\right) & \text{, otherwise} \end{cases} \tag{7.11}$$

Using the above equation for the common voltage, we scale the CPU and GPU power to a new hardware configuration using a V^2f model, as shown in Equations 7.12 and 7.13.

$$P_{CPU}(V_{CPU},\ f_{CPU}) = P_{CPU,\ baseline} \times \left(\frac{V_{CPU}}{V_{CPU,baseline}}\right)^2 \times \frac{f_{CPU}}{f_{CPU,baseline}} \tag{7.12}$$

$$P_{GPU}(V_{NB\_GPU},\ f_{NB},\ f_{GPU}\ f_{CU}) = P_{GPU,\ baseline} \times \left(\frac{V_{NB\_GPU}}{V_{NB\_GPU,\ baseline}}\right)^2 \times \frac{f_{GPU}}{f_{GPU,baseline}} \times \frac{CU_{GPU}}{CU_{GPU,\ baseline}} \tag{7.13}$$

7.4.3 Profiled Baseline Performance

Our MPC framework uses the profiled baseline performance data to determine the search order and to predict the power and performance. When the application is encountered for the first time, we store the CPU and the GPU performance counters, and the power numbers for each dissimilar sub-phase. Within a phase, the GPGPU benchmark is iterated multiple times. We store the performance counters for the dissimilar kernels and for its first, middle and the last CPU phase from its first invocation. We use 32-bit floating point to store six performance counters, three additional 32-bit floating point values for the power numbers, one 32-bit floating point value for the sub-phase time, two bits for the nature of the overlap, and three bits for the baseline configuration. For a workload with K sub-phases, we store $K \times 41$ bytes of information.

7.4.4 Adaptive Horizon Generator

Our MPC-based power manager adaptively varies the horizon length to limit the net performance loss, which includes the MPC overhead time and performance losses or gains from the MPC decisions, to a slowdown factor of α.

For the first sub-phase, our MPC power manager selects a full horizon length. Since the MPC scheme only optimizes for a subset of sub-phases that appear before the current sub-phase in the search order, the effective horizon length for the first sub-phase is $\mathbb{Z}_1$. If T_{m1} is the MPC time taken by the CPU to determine the hardware setting for the first sub-phase after it has optimized $\mathbb{Z}_1$ future sub-phases, the MPC optimization time for each sub-phase is $\frac{T_{m_1}}{\mathbb{Z}_1}$. If Z is the full horizon length, and $\mathbb{Z}$ is the average effective horizon length across all

sub-phases, the effective horizon length for a horizon length of H_j can be linearly approximated to be $H_j\frac{\mathbb{Z}}{Z}$. This quantity, when multiplied by $\frac{T_{m_1}}{\mathbb{Z}_1}$ gives the estimated MPC optimization time for a horizon length of H_j.

To determine H_j, we attempt to limit the overall slowdown due to past MPC optimization overheads $\left(\sum_{k=1}^{j-1} T_{m_{i,k}}\right)$, past performance $\left(\sum_{k=1}^{j-1} T_{i,k}\right)$ and the baseline time of the upcoming j^{th} sub-phase $\left(T_{B_j}\right)$ with respect to the total baseline times of j sub-phases $\left(\sum_{k=1}^{j} T_{B_k}\right)$ to be less than $1+\alpha$. This is shown in Equation 7.14.

$$\frac{H_j\frac{\mathbb{Z}}{Z}\frac{T_{m_1}}{\mathbb{Z}_1} + \sum_{k=1}^{j-1}\left(T_{m_{i,k}} + T_{i,k}\right) + T_{B_j}}{\sum_{k=1}^{j} T_{B_k}} \leq 1+\alpha \tag{7.14}$$

$$\text{where } Z = \sum_{i=1}^{P} G_i \times (2N_i + 1)$$

We solve H_j to get:

$$H_j \leq \frac{Z}{\mathbb{Z}}\frac{\mathbb{Z}_1}{T_{m_1}}\left[(1+\alpha)\sum_{k=1}^{j} T_{B_k} - \sum_{k=1}^{j-1}\left(T_{m_{i,k}} + T_{i,k}\right)\right] \tag{7.15}$$

We further take the floor of H_j to create an integer value and bound it between 0 and Z. A horizon length of zero indicates that a fail-safe configuration should be selected. We provide the details of the fail-safe configuration in the next section.

Table 7.3: Heterogenous benchmarks studied in this chapter.

Benchmark	Sequence of SPEC-GPGPU Combination
w0	hmmer-NBody, mcf-NBody, hmmer-G_lbm, mcf-XSBench, hmmer-G_lbm
w1	hmmer-G_lbm, gamess-NBody, gamess-G_lbm, C_lbm-XSBench, C_lbm-G_lbm
w2	gamess-G_lbm, mcf-NBody, hmmer-G_lbm, gamess-NBody, mcf-NBody
w3	mcf-XSBench, gamess-G_lbm, mcf-XSBench, C_lbm-G_lbm, hmmer-NBody
w4	hmmer-NBody, mcf-NBody, gamess-NBody, C_lbm-XSBench, C_lbm-G_lbm
w5	gamess-XSBench, gamess-NBody, gamess-G_lbm, mcf-NBody, mcf-NBody
w6	hmmer-G_lbm, gamess-G_lbm, gamess-NBody, mcf-G_lbm, gamess-NBody
w7	gamess-XSBench, C_lbm-G_lbm, gamess-NBody, mcf-XSBench, hmmer-G_lbm
w8	C_lbm-XSBench, C_lbm-NBody, gamess-NBody, mcf-G_lbm, mcf-G_lbm
w9	mcf-G_lbm, C_lbm-NBody, mcf-XSBench, hmmer-G_lbm, mcf-NBody
w10	C_lbm-G_lbm, mcf-XSBench, gamess-G_lbm, C_lbm-XSBench, mcf-G_lbm
w11	gamess-NBody, gamess-G_lbm, hmmer-NBody, C_lbm-G_lbm, C_lbm-NBody
w12	mcf-NBody, gamess-NBody, hmmer-G_lbm, gamess-G_lbm, gamess-XSBench
w13	mcf-G_lbm, hmmer-G_lbm, hmmer-G_lbm, gamess-NBody, C_lbm-XSBench
w14	gamess-XSBench, C_lbm-NBody, hmmer-G_lbm, mcf-G_lbm, C_lbm-G_lbm

7.5 Experimental Setup

7.5.1 Heterogeneous Workloads

Our synthetic workloads are composed of five phases, each with an overlap of randomly selected SPEC CPU 2006 and GPGPU benchmarks from Table 7.1. Table 7.3 shows a list of our heterogeneous workloads composed of such phases.

In this chapter, we use the AMD A10-7850K APU. We implemented our MPC-based power management framework on the host CPU at the configuration [P5, NB0, DPM0, CU2]. The CPU runs MPC prior to the invocation of a sub-phase. When there is no overlapping GPGPU execution, our scheme sets this configuration until it notices a change in the memory intensity of the next

phase.

In order to simulate our approach, we collect performance and power traces of the individual and overlapped execution of the SPEC CPU 2006 and GPGPU benchmarks across 96 hardware configurations (four CPU DVFS states, two NB DVFS states, three GPU DVFS states and four ways to change the GPU CU count). The AMD A10-7850K APU has two dual-core CPUs. We run three threads of the SPEC benchmark, while the remaining core is used to run the GPGPU benchmark. We take the sum of the *Retired_Instr_User* counter value across the three cores running SPEC to capture the SPEC instruction count of a sub-phase. To compute the cycles-per-instruction (CPI) of the SPEC benchmark within a sub-phase, we also take the sum of *CPU_Clock_not_Halted* counter value across the three cores and divide it by the sum of *Retired_Instr_User* of the three cores. The CPI of the CPU phase of the GPGPU benchmark is calculated by dividing the *CPU_Clock_not_Halted* with *Retired_Instr_User* of the fourth core. We use AMD CodeXL to capture the GPU performance counters [4] and measure the CPU and GPU power from the APUs power management controller at 1ms intervals. The CPU performance counters are monitored by reading the internal APU registers at 1ms intervals.

Within a phase, the GPGPU benchmark is iterated multiple times to reflect the scenario of the CPU and the GPU being used simultaneously. In our evaluation, the number of times the GPGPU benchmark is iterated matches with that of AMD Turbo Core, which completely overlaps SPEC with the GPGPU calls. Our proposed state-of-the-art baseline policy may not overlap SPEC and GPGPU calls entirely as with AMD Turbo Core. As a result, it can result in SPEC finishing on the CPU earlier or later than the GPU.

7.5.2 Baseline Schemes

Our state-of-the-art baseline scheme is inspired by the policy of separating the memory-bound phases to reduce the memory interference and performing DVFS on them [113]. Motivated by this approach, our first proposed *separating* baseline scheme executes memory-bound SPEC and memory-bound GPGPU calls separately. During this time, it selects the highest memory DVFS states and the lowest CPU states. For other types of phases, our scheme executes them together while keeping the unused power states low. If either the CPU or GPU becomes idle, the corresponding hardware setting is changed to its lowest state. Table 7.4 shows the details of our rule-based policy followed by both *separating* and *non-separating* baselines.

When MPC is targeting the performance of the *separating* baseline, it attempts to match the instruction count of a subphase to that of an estimated instruction count of SPEC from the baseline configuration. This is done to avoid unnecessarily penalizing the current sub-phase because of the comparison between a short sub-phase with that of the entire execution of SPEC benchmark. As a result, MPC assumes that the SPEC instruction count is equally distributed across all its sub-phases overlapping with the GPGPU benchmark, and it also assumes that the SPEC benchmark always overlaps with the GPGPU benchmark in the future for accurately predicting the sub-phase execution time.

Contrary to the results of Merkel et al. [113], our analysis from Section 7.2 shows that complete separation of memory-bound phases does not result in maximum energy efficiency. As a result, we developed a *non-separating* baseline that does not separate memory-bound SPEC and memory-bound GPGPU calls. The *non-separating* approach configures the APU as shown in Table 7.4.

Table 7.4: Our proposed rule-based policy followed by both *separating* and *non-separating* baselines, which keeps the performance insensitive hardware states to its lowest setting. Refer to Table 6.1 for the voltage and frequency settings of the CPU, NB and the GPU.

Nature of Computation		**HW Configuration**
CPU	**GPU**	
Compute	Compute	P1, NB3, DPM4, CU8
	Memory	P1, NB0, DPM0, CU2
	Idle	P1, NB3, DPM0, CU2
Memory	Compute	P7, NB0, DPM4, CU8
	Memory	P7, NB0, DPM0, CU2
	Idle	P7, NB0, DPM0, CU2
Idle	Compute	P7, NB3, DPM4, CU8
	Memory	P7, NB0, DPM0, CU2

Table 7.5: Failsafe configuration selected by our MPC power manager.

CPU		**GPU**		**Hardware Configuration**
Status	**Type**	**Status**	**Type**	
Idle	–	Active	Any	P7, NB0, DPM4, CU8
Active	Any	Idle	–	P1, NB0, DPM0, CU2
Active	Memory	Kernel active	Any	P7, NB0, DPM4, CU8
Active	Compute	Kernel active	Any	P1, NB0, DPM4, CU8
Active	Memory	CPU portion active	Any	P7, NB0, DPM0, CU2
Active	Compute	CPU portion active	Any	P1, NB0, DPM0, CU2

Upon encountering the application for the first time, we run our proposed baseline, capture the performance counters and set that performance as a target for our MPC power manager. MPC predicts the performance and power behavior of future sub-phases and uses that information to select the hardware configuration for the upcoming sub-phase such that it performs better than the baseline performance, and also processes the same amount of SPEC instructions. If MPC is unable to find a configuration, it switches to an empirically determined fail-safe configuration, shown in Table 7.5. Our adaptive MPC scheme varies

the horizon length dynamically to limit the net performance loss to be less than $\alpha = 5\%$.

7.6 Results

In this section, we present the energy savings and performance impact of MPC with respect to the *separating* and *non-separating* baselines. All of our results use the prediction model, unless otherwise mentioned. We first show the results for the phases and then present the gains for our benchmarks. Lastly, we compare the performance using our prediction model with a perfect prediction model.

7.6.1 Energy-Performance Gains

Phase Comprised of SPEC-GPGPU Overlap

Figure 7.11 shows the energy savings and performance impact of MPC with full horizon for the phases composed of overlapping SPEC and GPGPU benchmarks with respect to the *separating* and *non-separating* baselines. The results presented here do not include the MPC overheads. Our later results for the heterogeneous workloads composed of these phases include the associated overheads. Furthermore, we do not show the overlap of *hmmer* and *XSBench* because we missed collecting its traces for one particular configuration.

Phases comprised of memory-bound SPEC and memory-bound GPGPU benchmarks, such as *C_lbm* and *G_lbm*, and *mcf* and *G_lbm*, show more than 30% energy savings and greater than 35% performance improvement with respect to

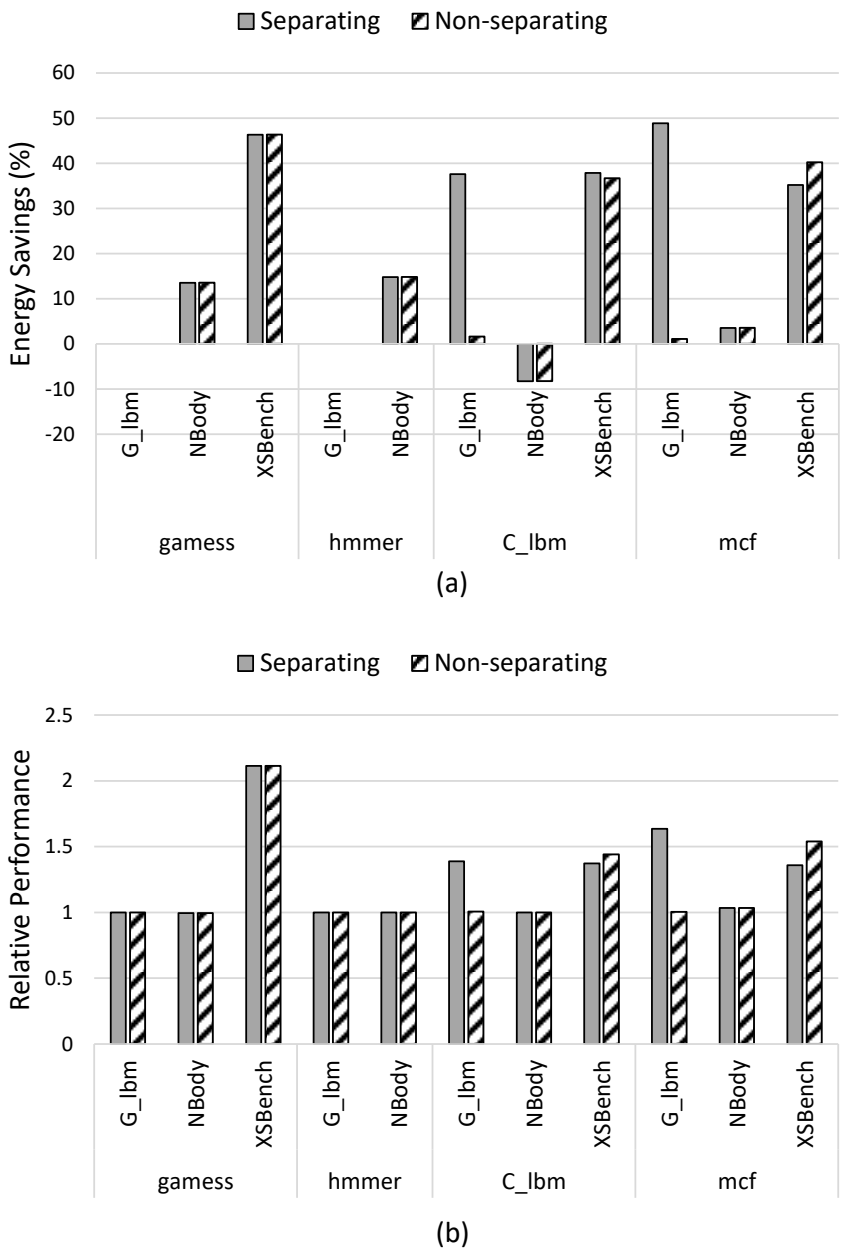


Figure 7.11: Full MPC (a) energy savings and (b) relative performance (without overheads) for phases over the *separating* and *non-separating* baselines.

the *separating* baseline. This is because the baseline separates these phases and hence there is a significant performance headroom for MPC to reduce energy. With respect to the *non-separating* baseline, the savings reduces because of less performance headroom. However, when *XSBench* overlaps with *mcf* or *C_lbm*, we observe an improvement in performance and for *C_lbm*, a slight increase in

energy savings. This is because MPC with the *separating* baseline assumes the SPEC instruction count to be equally distributed among its phases, and that the SPEC and GPGPU overlaps persist throughout the workload. As a result, it does not permit accelerating the kernels in order to meet the SPEC instruction count constraint, and therefore shows relatively smaller performance improvement. For the rest of the benchmarks, results match for both *separating* and *non-separating* baselines since both the baseline policies are identical.

For phases involving *G_lbm*, we achieve negligible energy savings irrespective of the SPEC benchmark. This is because *G_lbm* shows performance peak behavior [107], which our performance model is unable to capture from just one baseline performance sample.

For the overlap of compute-bound SPEC and compute-bound GPGPU benchmarks, such as *hmmer* and *NBody*, and *gamess* and *NBody*, MPC achieves 14.8% and 13.5% energy savings, respectively, with no performance impact. With respect to the baseline configuration of [P1, NB3, DPM4, CU8], MPC intelligently lowers the GPU configuration of the CPU phase of the GPGPU benchmark, thereby lowering the energy without impacting its performance. For *NBody*, the GPU kernels constitute more than 85% of the overall execution time, and therefore, the energy savings comes from the remaining 15% of the GPGPU benchmark.

For memory-bound SPEC and compute-bound GPGPU overlaps, such as *mcf* and *NBody*, and *C_lbm* and *NBody*, MPC matches the performance for both but achieves 3.5% energy savings for the former and loses 8.3% energy for the latter. This is because the baseline configures the hardware at [P7, NB0, DPM4, CU8], which is energy-efficient. MPC attempts to reduce energy by infrequently

lowering the NB DVFS state. The benchmark *mcf* is less memory sensitive than *C_lbm*. The former typically loses around 35% of performance when the processor is configured at NB3, while *G_lbm* loses half of its performance. Because of these characteristics, *mcf* and *NBody* lose less performance, which it recovers from slightly increasing the DVFS state of the CPU to speed up the performance of *mcf*. However, for *C_lbm* and *NBody*, MPC significantly degrades the performance of *C_lbm*, which it is unable to recover later on. MPC, therefore, chooses the fail-safe configuration and meets the performance target at the end, but loses energy. Even a perfect prediction model shows an energy loss of 4% for the phase *C_lbm* and *NBody*.

For the overlap of compute-bound SPEC and memory-bound GPGPU benchmarks, such as *gamess* and *XSBench*, MPC achieves an energy savings of 46.4% with a performance improvement of 2.1×. The baseline policy selects [P1, NB0, DPM0, CU2] when *gamess* and *XSBench* are overlapping and then [P7, NB0, DPM0, CU2] for the remaining portion of *XSBench*. MPC looks ahead in the future, selects higher GPU DVFS states to accelerate future *XSBench* kernels that are not overlapping with *gamess*, and uses that performance gain to lower the CPU DVFS states, thereby stretching the execution time of the SPEC benchmark. The overall effect of doing so increases the performance with respect to the baseline and simultaneously reduces energy. MPC with a horizon length of one is unable to foresee the future, and therefore can only accelerate the GPU kernels not overlapping with the SPEC benchmark, and therefore achieves an energy savings of only 19.1%.

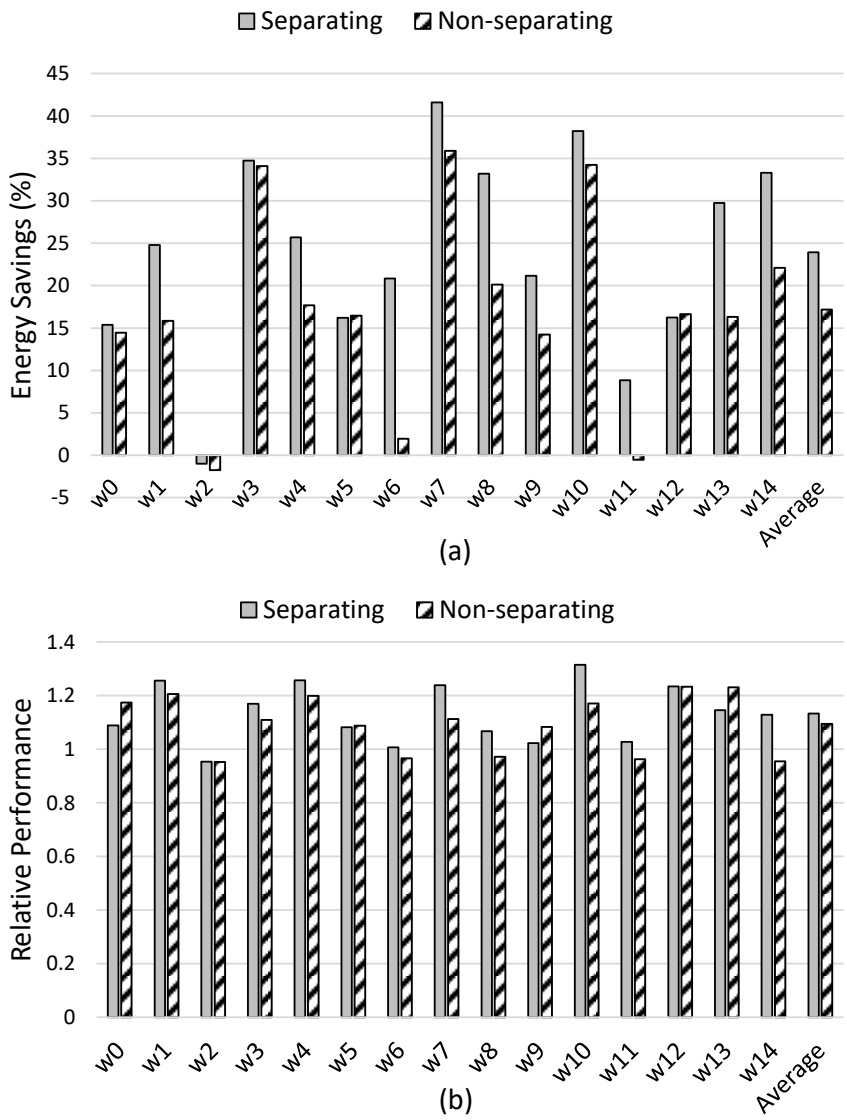


Figure 7.12: Adaptive MPC (a) energy savings and (b) relative performance (with overheads) for the different workloads compared to the *separating* and *non-separating* baselines.

Heterogeneous Workloads

Figure 7.12 shows the energy savings and performance impact of MPC for the heterogeneous workloads using the adaptive horizon and 2-level heuristic. We present our results against *separating* and *non-separating* baselines and the results include the associated MPC overheads. We quantify the associated overheads

later in Section 7.6.2.

With respect to the *separating* baseline, MPC achieves an average energy savings of 24% while improving performance by 13.3%. These include an energy overhead of 1.8% and a performance overhead of 2.6%. On the other hand, with respect to the *non-separating* baseline, MPC achieves 17.2% energy savings, and improves performance by 9.4%. The MPC overheads with respect to the *non-separating* baseline are 1.5% for energy and 2.4% for performance.

As shown in the previous section, the individual energy savings and performance impact for each SPEC-GPGPU phase depends on the nature of the overlap. Therefore, the relative energy savings of a workload not only depends on the distribution of phases but also on the sequence of phases within an application. For instance, the workload *w2* degrades energy compared to both the baselines since it does not have any overlap of memory-bound SPEC and memory-bound GPGPU benchmarks. Contrary to this, workloads *w3* and *w10* show more than 30% energy savings because they have 80% overlap of memory-bound SPEC and memory-bound GPGPU benchmarks.

Furthermore, if a phase with memory-memory overlap appears in the beginning of the workload, the performance gain from that phase gives more headroom for the subsequent phases. Similarly, the performance loss from initial phases causes MPC in subsequent phases to prefer high-power configurations to mitigate the performance loss. Workloads *w7* and *w14* contain two memory-bound phases and both start off with *gamess* and *XSBench*, which has greater potential of energy savings and performance gain, as shown in Figure 7.11. However, *w7* shows higher energy savings than *w14* because *gamess* and *XSBench* is immediately followed by *C_lbm* and *G_lbm* which adds more performance head-

room that is used later by the subsequent phases. In *w14*, the memory-bound phases appear towards the end.

The workload *w11* degrades energy by 0.5% with respect to the *non-separating* baseline, while it reduces energy by 8.8% over the *separating* baseline. This is because the workload contains an overlap of *C_lbm* and *G_lbm*, which are both memory-bound. The *separating* baseline shows energy savings because of extra performance headroom, while the *non-separating* baseline consumes more energy because of our prediction model's inability to accurately predict the performance peak behavior of *G_lbm*. Similar reasons apply for the workload *w6*, except for the fact that MPC achieves a reduced energy savings of 2% with respect to the *non-separating* baseline.

7.6.2 Comparing the MPC Heuristics

Figures 7.13(a) and (b) show the energy savings and relative performance of adaptive MPC running different MPC search heuristics with respect to the *separating* baseline. The energy savings and relative performance include the associated MPC overheads. The overheads are shown separately in Figures 7.13(c) and (d). Figure 7.14 presents the results with respect to the *non-separating* baseline. For the results in this section, all the MPC algorithms were run with full horizon length.

Including the overheads, MPC using the *2-level* heuristic achieves 29.3% energy savings while improving performance by 2.7% when the baseline is the *separating* policy. On the other hand, the *coarse* heuristic reduces energy by 21.1% and improves performance by 8.5%, while *flat* achieves 23.6% energy savings

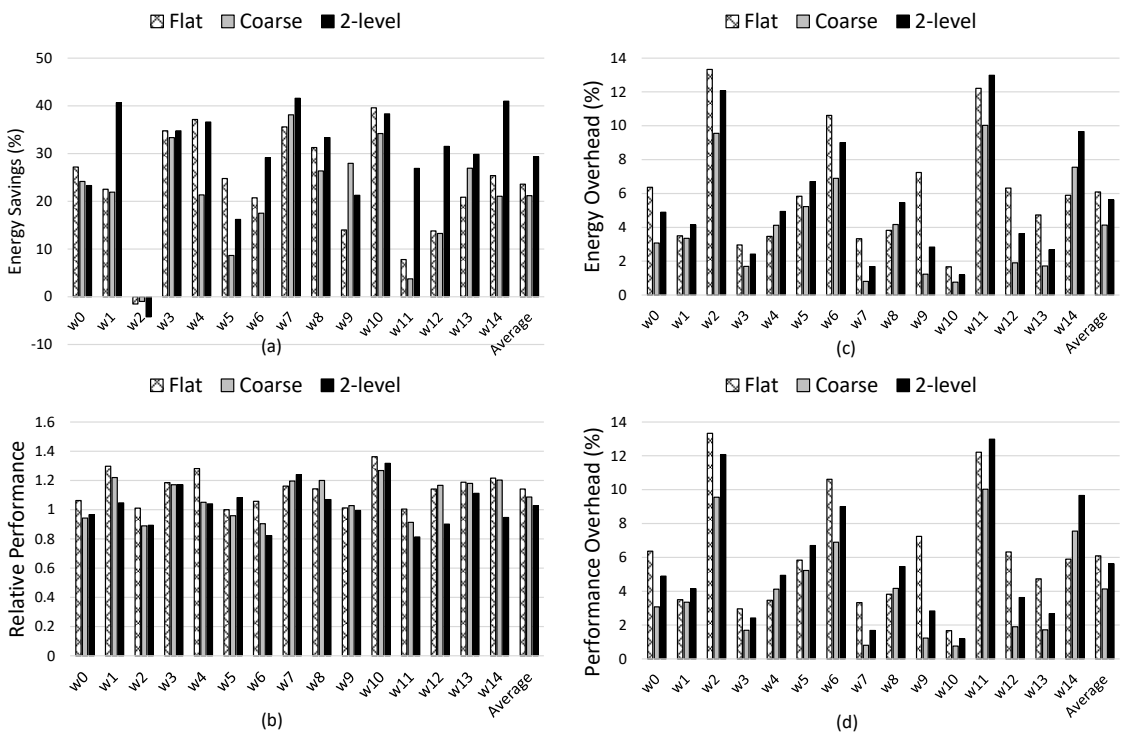


Figure 7.13: Full MPC (a) energy savings, (b) relative performance, (c) energy overhead and (d) performance overhead for different MPC search heuristics, relative to the *separating* baseline.

with a performance improvement of 14.1%. With respect to the *non-separating* baseline, the *2-level* heuristic achieves 22.2% energy savings while degrading performance by 5.2%. Without the overheads, the *2-level* heuristic meets the performance target. The *coarse* heuristic, however reduces energy by 15.5% and improves performance by 4.7%, while *flat* reduces energy by 12.3% and increases performance similar to the *coarse* heuristic.

In terms of average overheads, the *coarse* heuristic shows the least energy and performance overheads for both baselines. This is because the *coarse* heuristic has shorter future visibility since it orders the sub-phases sequentially within the slowest phase, thereby resulting in a shorter distance between the current and the future sub-phases. The *flat* heuristic has the largest overheads because it orders the sub-phases from the whole workload in terms of their decreasing

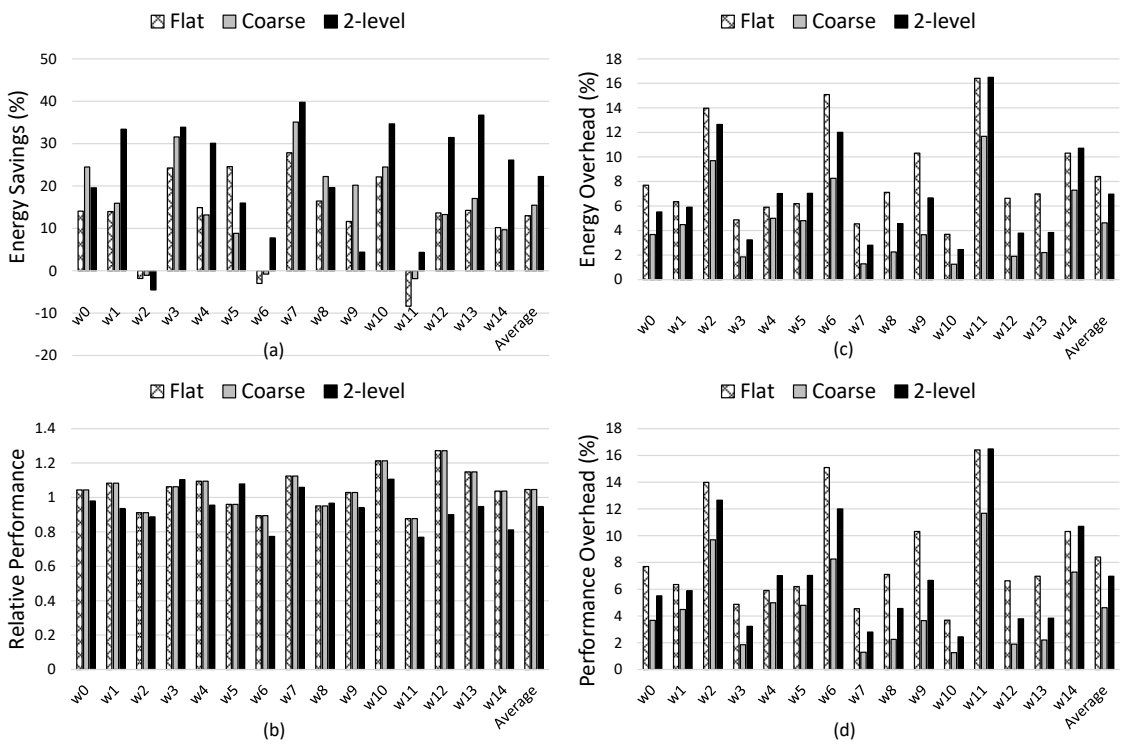


Figure 7.14: Full MPC (a) energy savings, (b) relative performance, (c) energy overhead and (d) performance overhead for different MPC search heuristics, relative to the *non-separating* baseline.

execution time. As a result, the future sub-phase does not remain concentrated in the slowest phase and therefore, the MPC algorithm has to look way ahead in the future. The *2-level* heuristic shows in-between overheads compared to *flat* and the *coarse* heuristics because it orders the slowest sub-phase within the slowest phase. As a result, it avoids looking far off in the future but also looks beyond the first sub-phases of the slowest phase.

For the individual workloads, we observe that the overheads of the *2-level* heuristic are greater than *flat* for the workloads *w1*, *w4*, *w5*, *w8*, *w11* and *w14* when compared to the *separating* baseline (Figures 7.13(c) and (d)). This is because the slowest phase involving *mcf* and *C_lbm* benchmarks appear toward the end of the workloads. Similar reason apply for the workloads *w4*, *w5*, *w11*

and *w14* with respect to the *non-separating* baseline. The *coarse* heuristic as well shows larger overheads with respect to the *flat* heuristic for the workloads *w4* and *w14* with respect to the *separating* baseline because of similar reasons.

Overall, we observe that the *2-level* heuristic achieves higher energy savings because it prioritizes the slowest phase and the slowest sub-phases within that phase. As a result, it dedicates the power resources to those sub-phases that most significantly contribute to the overall performance. The *coarse* heuristic, while prioritizing slower phases and thereby benefitting from the smaller overheads, achieves lower savings because it does not consider the performance differences of the sub-phases within a phase. The *flat* heuristic, despite using a longer future horizon, treats all the phases equally, and therefore shows lower energy savings along with higher overheads.

7.6.3 Impact of Horizon Length

Figures 7.15(a) and (b) show the MPC energy savings and relative performance for different horizon lengths with respect to the *separating* baseline. The energy savings and relative performance includes the associated MPC overheads, which are shown separately in Figures 7.15(c) and (d). Figure 7.16 compares the three MPC policies for different horizon lengths with respect to the *non-separating* baseline. Here, we compare three MPC polices: *MPC_1* using a horizon length of one; *adaptive_MPC* that adaptively varies the horizon length based on MPC optimization time and ongoing application performance; and *full_MPC* that looks into the entire future. Each of the proposed policies employ the 2-level search heuristic.

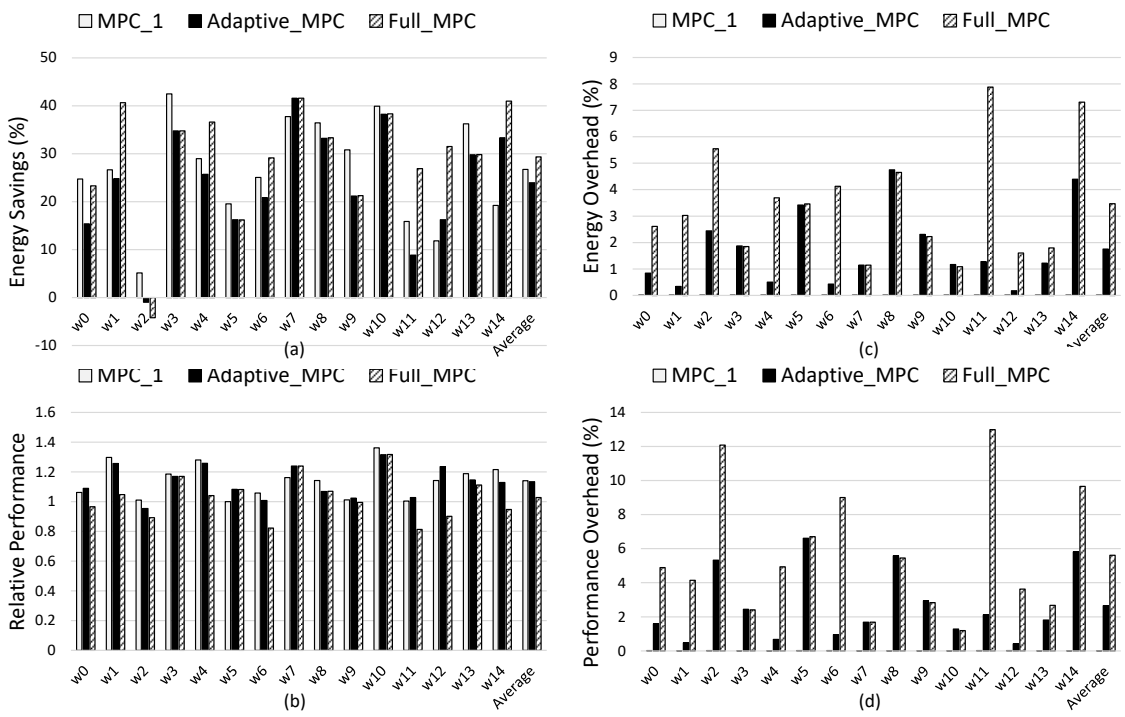


Figure 7.15: MPC (a) energy savings, (b) relative performance, (c) energy overhead and (d) performance overhead for varying horizon lengths, relative to the *separating* baseline.

With respect to the *separating* baseline, *full_MPC* improves performance by 2.7% after an overhead of 5.6% is included. Energy-wise, *full_MPC* reduces energy by 29.3% including an overhead of 3.5%. *MPC_1*, however, reduces the energy by 26.7%, thereby falling behind *Full_MPC*. In terms of performance, *MPC_1* speeds up the workloads by 14.1%. *Adaptive_MPC* trades off energy for higher performance and reduces energy by 24% with an overhead of 1.7%. Performance-wise, it speeds up the workloads by 13% after a performance overhead of 2.6% is included.

With respect to the *non-separating* baseline, shown in Figures 7.16(a) and (b), *MPC_1* achieves an energy savings of 19.3% while improving performance by 7% with negligible overheads. *Full_MPC*, however, achieves a greater energy savings of 22.2%, which includes an energy overhead of 4.2%. Performance-

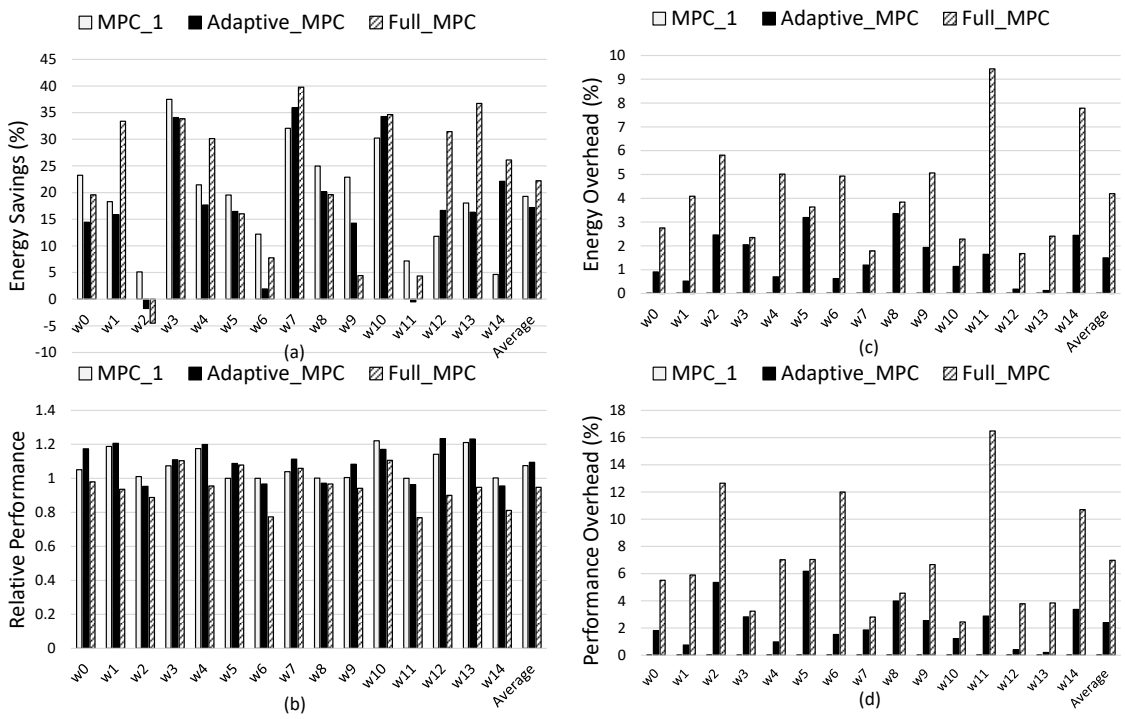


Figure 7.16: MPC (a) energy savings, (b) relative performance, (c) energy overhead and (d) performance overhead for varying horizon lengths, relative to the *non-separating* baseline.

wise, *full_MPC* degrades performance by 5% because of a performance overhead of 6.8%. *Adaptive_MPC*, similar to the *separating* baseline, trades off energy for higher performance, and achieves an energy savings of 17.2% with a overhead of 1.5%, and improves performance by 9.4% with an overhead of 2.4%.

Common to both baselines, for certain workloads, such as *w6*, *w12* and *w14*, MPC trades off performance for better energy savings by looking ahead in the future. However, workloads such as *w2*, *w3*, *w5*, *w8* and *w9* do not benefit from a longer horizon. This is because looking far ahead creates negative performance credit for these workloads, which causes MPC to select the high-power configuration, thereby degrading energy efficiency. The *adaptive_MPC* policy does worse than *MPC_1* and *MPC_full*, for workloads such as *w6* and *w11*. This is because of limited future visibility, which creates negative performance credit,

thereby impacting the current sub-phase. The *MPC_1* policy does not accumulate negative credit from the future and therefore achieves better savings, whereas *full_MPC* looks farther ahead and includes the credit from the far-away sub-phases to reduce the performance penalty.

7.6.4 Comparison with Perfect Model

Figure 7.17 presents the impact of prediction inaccuracy on MPC's energy and performance results with respect to the *separating* baseline, while Figure 7.18 is in reference to the *non-separating* baseline. In this section, we consider MPC with the 2-level heuristic and full horizon to eliminate the effects of varying horizon length. Our results also do not include the associated overheads for a fair comparison against the perfect model.

With respect to the *separating* baseline, MPC using our prediction model reduces energy by 32.8% while improving performance by 8.3%. A perfect model saves 38.8% energy, while improving performance by 28%. When the baseline is *non-separating*, MPC with prediction achieves 26.4% savings with no performance impact, while the perfect prediction model saves 29.2% energy and improves performance by 5%.

The reason behind the difference in the savings between the prediction and the perfect model stems from the inaccuracy of our prediction model. Table 7.6 shows the prediction accuracy for all the phases. The power accuracy is expressed in terms of the Spearman rank correlation coefficient [14]. This is because our problem formulation does not use the energy in its constraints and a proxy function of the power having the same monotonicity as the actual power

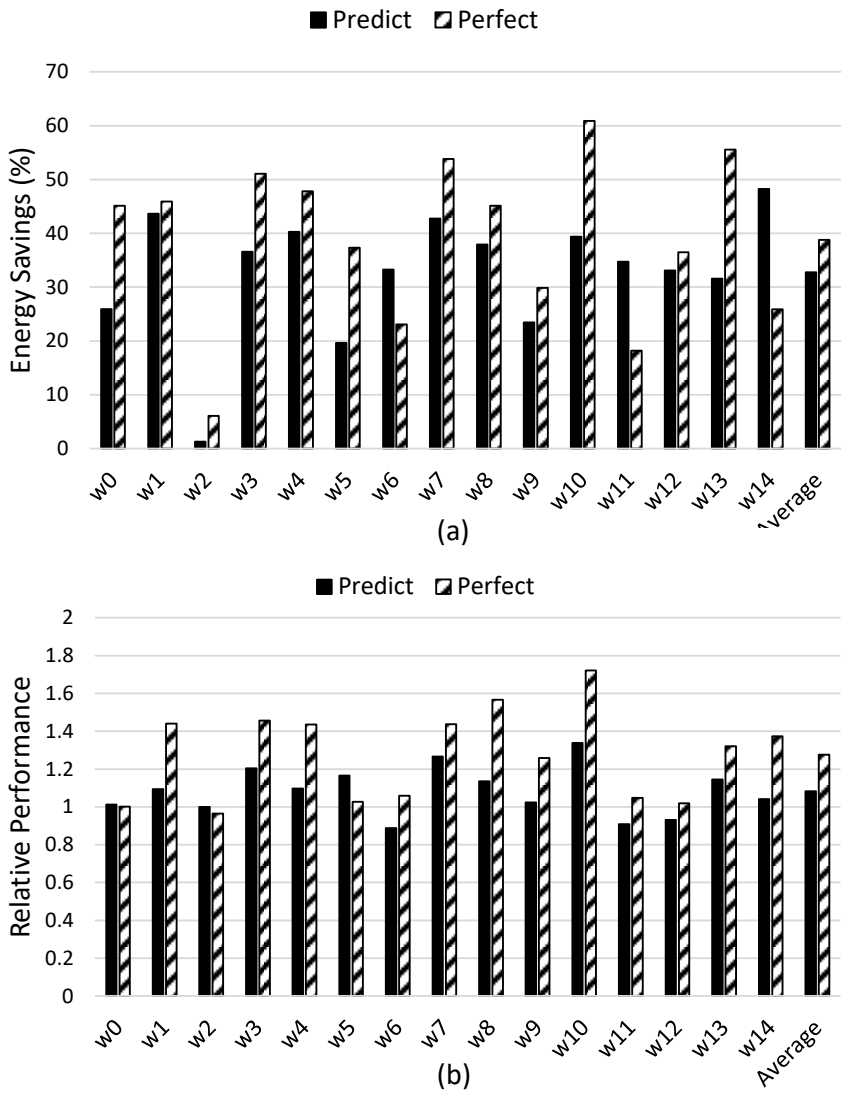


Figure 7.17: Comparison of MPC (a) energy savings, and (b) relative performance using the prediction model and a perfectly accurate model, with respect to the *separating* baseline.

function is sufficient for our MPC framework.

As shown in Table 7.6, our prediction models are fairly accurate for phases involving *NBody*. This is because *NBody* is a compute-bound benchmark that scales linearly with the GPU DVFS states and the GPU CU count. As a result, our GPU prediction models that scale the kernel performance based on the relative change in the GPU CU count achieve good accuracy for *NBody*. However,

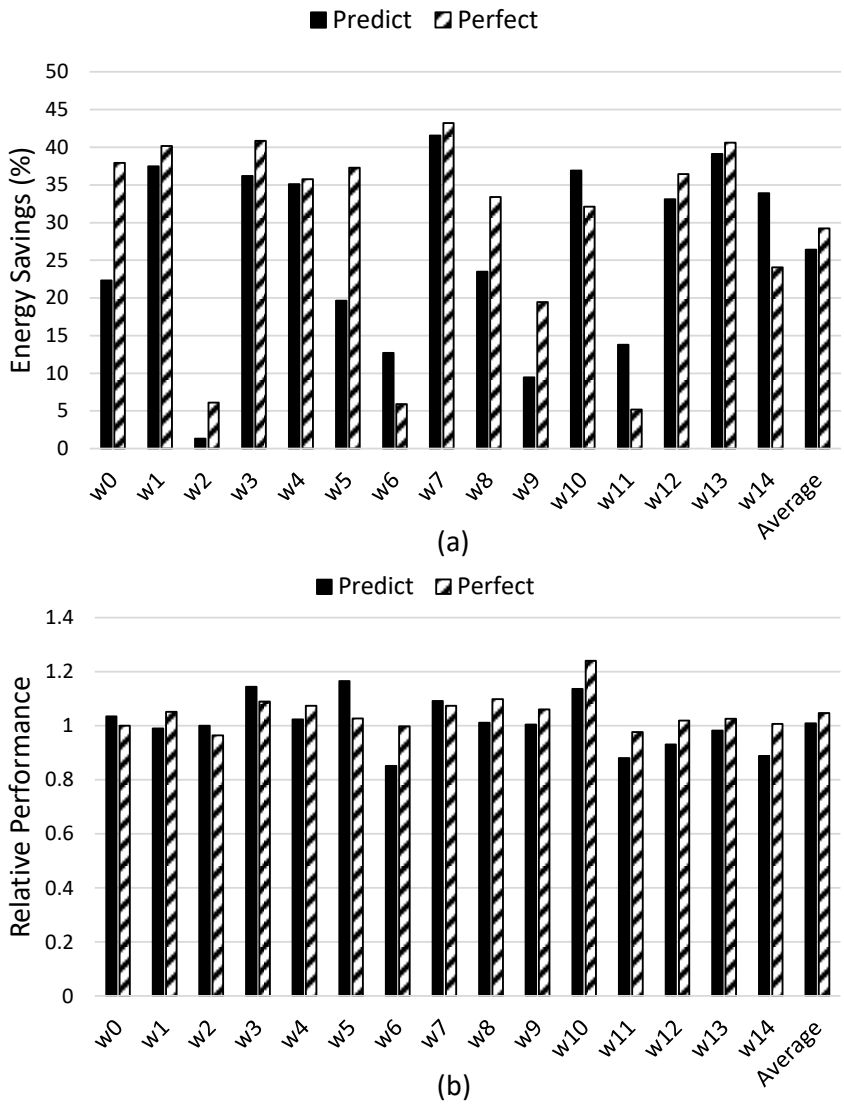


Figure 7.18: Comparison of MPC (a) energy savings, and (b) relative performance using the prediction model and a perfectly accurate model, with respect to the *non-separating* baseline.

for memory-bound GPGPU benchmarks, such as *XSBench* and *G_lbm*, there is no linear relationship. Moreover, the *G_lbm* benchmark exhibits performance peaks, which our prediction model has difficulty handling based on one baseline performance reference. Furthermore, our prediction model does not consider the impact of sharing the memory bandwidth between the CPU and the GPU, which further affects the accuracy.

Table 7.6: Prediction accuracy.

SPEC	GPGPU	% Accuracy		Spearman Correlation Coefficient
		Sub-phase time	Instruction Count	Power
gamess	G_lbm	77.2	76.2	0.67
	NBody	95.4	90.5	0.90
	XSBench	73.0	68.9	0.58
hmmer	G_lbm	77.3	77.0	0.66
	NBody	91.4	88.2	0.90
C_lbm	G_lbm	74.7	72.5	0.49
	NBody	96.5	73.3	0.87
	XSBench	73.0	77.7	0.51
mcf	G_lbm	76.1	75.3	0.51
	NBody	96.4	87.3	0.82
	XSBench	75.5	80.8	0.57

Similarly, we observe higher prediction accuracy for the instruction count of either the SPEC or the CPU portion of the GPGPU benchmark whenever *NBody* is involved. This is because the prediction model to predict the instruction count is based on the PPEP model. The original PPEP model considers the variation in the CPU DVFS states while keeping the NB DVFS states fixed. Our extension of the PPEP model to predict the CPI based on normalized NB frequency works well for *NBody* because of its smaller *mem_intensity* factor. However, for the memory-bound CPU sub-phases, it does not estimate CPI accurately. Furthermore, we observe variation in the SPEC instruction as the CPU or the NB DVFS states vary. This is because the performance counters also include the instruction counts of the tool used in capturing the traces. At lower DVFS states, we observe an increase in the instruction count because our tool busy-waits at the synchronization loops for a longer time.

For power, we observe higher accuracy for phases with *NBody* because the GPU power scales linearly with the increase in CU count because of high activity factor. However, the increase in GPU power with increase in the CU count for the memory-bound GPGPU benchmarks scales poorly. This is because with an increase in CU count, memory bandwidth can become the bottleneck with an increase in the memory requests, which results in stalling both the CPU and the GPU. Furthermore, our power does not isolate the static power or consider the variation in CPU static power due to increase in the GPU power states, which further impacts the accuracy.

However, despite these inaccuracies, our MPC based power manager achieves close to the savings of the perfect model because it looks into the future sub-phases and takes continuous performance feedback to mitigate the effects of prediction inaccuracy.

7.7 Conclusion

In this chapter, we extended our MPC-based power management architecture for heterogeneous applications using the CPU and the GPU concurrently. Inspired by real-world heterogeneous applications, such as *gromacs* and the AMD A10-7850K's implementation of *linpack*, we create synthetic heterogeneous workloads running SPEC on the CPU and GPGPU benchmarks on the GPU. Our MPC power manager determines appropriate hardware settings for the future portions of the applications and uses the performance credit to configure the hardware for the upcoming application sub-phase. We examine three different MPC search heuristics, and observe that the 2-level heuristic balances

the overheads with respect to the *flat* and the *coarse* heuristics and achieves more energy savings. We compare our proposed policy with two baselines: a state-of-the-art policy that separates the memory-bound portions, and an improvement over this policy that runs the memory-bound portions together. Compared to the first baseline, our approach reduces energy by 24% while increasing performance by 13.3%. With respect to the second baseline, MPC achieves 17.2% energy savings with a performance improvement of 9.4%.

CHAPTER 8

FUTURE WORK

There are a number of ways this dissertation research could be expanded. In this chapter, we discuss ideas for further improving this research including new directions of *proactive* methods to further improve the operational performance of smart buildings and heterogeneous architectures.

8.1 Energy Efficient Meeting Assignment

In Chapters 3 and 4, we proposed an energy model to efficiently assign meetings to rooms to reduce the building HVAC energy. Our formulation assumes *a priori* knowledge of the meeting schedule. In reality, meeting schedules vary dynamically because of last minute cancellations or rescheduling, and weather conditions can change without a forecast. Under these variable circumstances, future work can dynamically assign meetings to maximize energy savings while limiting occupant inconvenience from rescheduling the meetings. Similarly, our meeting assignment problem could be further extended to energy efficient fusion and fission of group offices. In academic or corporate buildings, group offices are arranged in an ad hoc fashion with grouping people either by common interests or common managers. However, individual occupants may have different arrival and departure patterns. Grouping people with common patterns of occupancy to suitable rooms along with the weather factors is an interesting research problem; we believe intuitions from our proposed energy model can be applied to such problems.

With regards to meeting assignment, our current energy model studies the impact of solar factors for a two-room building, with rooms directly facing or opposite to the sun. Future research can exploit indirect orientation toward the sun with buildings of more than one room. Furthermore, the occupancy factor (γ) considered in our model depends on the active energy of a room with full occupancy. We assumed an occupant density of 12 square feet of area per person for a theatre style room [103]. However, in our simulation, an occupancy of such a high density can produce heating from the occupants on certain days, resulting in zero heating energy. This impacts the occupancy factor, and as a result, our proposed model loses energy savings. Further research is needed to examine the optimum number of occupants to avoid such a condition, and to study the impact of weather on the occupancy factor.

8.2 Dynamic HVAC Energy and Occupant Comfort Optimization

In Chapter 5, we proposed Model Predictive Control (MPC) to dynamically balance the HVAC energy and occupant comfort. Our current framework is simulated for a one room building and a probabilistic model predicting binary occupancy. Future work can extend this framework to multiple rooms with variable occupancy. Furthermore, our current framework assumes *a priori* knowledge of building dynamics and precise weather forecasts. In reality, applying this framework to an actual building requires accurate modeling of the building dynamics in a simulator or using building energy models. Future work could study the impact of model inaccuracies and that of inaccuracy in the weather

forecast on the energy savings and occupant comfort.

Furthermore, with respect to the implementation of our MPC framework, our approach does not provide a way to strictly avoid violating the occupant comfort requirement. While providing strict guarantees can severely degrade the energy savings, it would be worthwhile investigating dynamically varying the strictness based on runtime occupant discomfort. Future work can explore a chance-constrained formulation to analyze the tradeoff between energy and discomfort.

8.3 Dynamic Power Management of Heterogeneous Architectures

In Chapters 6 and 7, we apply MPC to dynamically balance system-level application energy of heterogeneous processors without impacting its performance. Our method uses past performance and anticipated gains or losses from the future to dynamically tune its current decision. MPC can as well be applied to areas of application scheduling or thermal management both at the processor and cluster levels by knowing the anticipated application demand or thermal characteristics of the application and that of the compute nodes.

In MPC, future visibility requires knowledge of the upcoming execution pattern of the application, which requires the application should be run at least once. Our proposed MPC power management framework uses profiled performance data for future visibility. Future work can avoid profiling by dynamically extrapolating the future execution pattern based on past execution history.

Furthermore, our MPC framework for GPGPU applications uses a Random Forest based performance predictor, which is trained on numerous GPGPU benchmarks across multiple hardware configurations. In addition to the produced model being machine specific, our produced Random Forest model has a large depth, which makes it difficult to implement on a micro-controller. Motivated by these limitations in Chapter 6, we developed a different approach to predict power and performance for MPC-based power management of heterogeneous applications. Instead of using machine learning models trained on numerous benchmarks, we developed analytical models based on PPEP [151] and Harmonia [128]. These models, even though machine independent, forecast power and performance based on the profiled data and captured performance counters. Our current model cannot forecast the cross-coupling effects of performance and power from one baseline data. These effects can include the impact on the SPEC performance due to increased memory pressure from high GPU power states or the increase in the static CPU power due to high GPU DVFS settings. Future work can better balance the flexibility of a machine independent model and the benefits of an offline trained model by employing a low-overhead online model that learns during execution.

CHAPTER 9

CONCLUSIONS

The increasing demand for improved operational performance along with depleting energy resources demands new approaches to improve the energy efficiency of smart buildings and computer systems. Conventional energy management techniques have been either reactive or locally predictive at best. Such schemes often underperform, by either failing to meet the desired performance target or consuming excess energy.

In this dissertation, we propose proactive power management techniques suitably adapted for Heating, Ventilation, and Air-Conditioning (HVAC) in smart buildings, and for dynamic power management of compute servers. We show how proactive power management techniques incorporating future visibility, continuous feedback from past decisions, and intelligent variation of system inputs can achieve significant energy savings and performance improvements while being practically deployed in these two domains.

For building energy management, we first propose to automatically assign meetings to rooms to reduce overall energy consumption. By characterizing the building energy behavior, we derive an HVAC energy model for meeting assignment. Our model considers rooms with different capacities, and meetings with varying occupancies, gaps and conflicts. Using this energy model, we propose several assignment algorithms, and analyze their optimality and scalability. We also characterize how different factors impact energy savings, when it is worthwhile to use complex assignment algorithms, and when simpler methods suffice. We further extend this model to understand how solar factors impact the meeting assignment and its associated energy savings. We develop a

methodology to model the energy difference between the south-facing and the north-facing rooms with the weather parameters, and include this factor in our energy model. Our solar-aware energy model intelligently schedules meetings to different rooms based on weather forecasts.

We next apply Model Predictive Control (MPC) to dynamically balance HVAC energy consumption and occupant comfort. Our MPC framework adaptively balances the energy consumption and occupant comfort by using recent discomfort history and a probabilistic model to predict the upcoming occupancy. We simulate both regular and irregular occupancy profiles, and achieve high energy efficiency while operating within a specified discomfort target.

We then propose to apply MPC to dynamically reduce the energy consumption of General Purpose Graphics Processing Unit (GPGPU) applications without impacting performance. Our approach proactively configures the hardware states based on recent execution history, the pattern of upcoming kernels, and the predicted behavior of those kernels, and achieves high energy savings with negligible performance impact. Our scheme achieves low energy and performance overheads by adaptively varying the amount of future visibility, thereby making it a practical runtime scheme for servers.

Finally, we propose to use MPC to increase the energy efficiency of heterogeneous applications using the CPU and the GPU concurrently. Our scheme uses the past performance, looks into the future time windows of the application, and adaptively varies its future visibility based on real-time MPC overheads to appropriately configure the hardware. We explore the tradeoff between future visibility and computational overhead, and demonstrate significant energy savings over state-of-the-art power management approaches.

BIBLIOGRAPHY

[1] 2010 Buildings Energy Data Book, US Department of Energy, Chapter 1. http://web.archive.org/web/20130214024505/http://buildingsdatabook.eren.doe.gov/ChapterIntro1.aspx. Accessed: 03/15/2012.

[2] APP SDK - A Complete Development Platform. http://developer.amd.com/tools-and-sdks/opencl-zone/amd-accelerated-parallel-processing-app-sdk/.

[3] clFinish(). https://www.khronos.org/registry/OpenCL/sdk/1.0/docs/man/xhtml/clFinish.html.

[4] CodeXL - Powerful Debugging, Profiling & Analysis. http://developer.amd.com/tools-and-sdks/opencl-zone/codexl/.

[5] CUDA C Programming Guide. http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#abstract.

[6] EnergyPlus Energy Simulation Software. http://apps1.eere.energy.gov/buildings/energyplus/.

[7] Executive Order – Creating a National Strategic Computing Initiative. https://www.whitehouse.gov/the-press-office/2015/07/29/executive-order-creating-national-strategic-computing-initiative. Accessed: 08/16/2015.

[8] Exhaustion of Fossil Fuels. http://www.terra-symbiosis.org/EN_epuisement-fossiles.pdf. Accessed: 08/16/2015.

[9] HSA Foundation. http://www.hsafoundation.com/.

[10] PHORONIX TEST SUITE. http://www.phoronix-test-suite.com/.

[11] Trimble Sketchup. http://sketchup.google.com.

[12] What is Heterogeneous System Architecture (HSA)? http://developer.amd.com/resources/heterogeneous-computing/what-is-heterogeneous-system-architecture-hsa/.

[13] XSBench: The Monte Carlo Macroscopic Cross Section Lookup Benchmark. https://github.com/ANL-CESAR/XSBench/.

[14] *Spearman Rank Correlation Coefficient*, pages 502–505. Springer New York, New York, NY, 2008.

[15] ASHRAE Standard 62.1-2010: Ventilation for Acceptable Indoor Air Quality. *American Society of Heating, Refrigerating, and Air-Conditioning Engineers, Inc.*, 2010.

[16] 2010 Buildings Energy Data Book, US Department of Energy, Table 3.1.5. http://buildingsdatabook.eren.doe.gov/docs/DataBooks/2010_BEDB.pdf, Mar. 2011. Accessed: 03/15/2012.

[17] IBM ILOG CPLEX Optimizer. http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/, Dec. 2013.

[18] Americaś Data Centers Are Wasting Huge Amounts of Energy. *Natural Resources Defense Council*, 2014.

[19] Top Ten Exascale Research Challenges. *U.S. Department of Energy*, 2014.

[20] Advanced Micro Devices, Inc. *BIOS and Kernel Developer's Guide (BKDG) for AMD Family 15h Models 30h-3Fh Processors*, February 2015.

[21] Y. Agarwal, B. Balaji, R. Gupta, J. Lyles, M. Wei, and T. Weng. Occupancy-Driven Energy Management for Smart Building Automation. In *Proc. of the ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Building (BuildSys)*, pages 1–6, New York, NY, USA, 2010.

[22] S. Ari, I. A. Cosden, H. Khalifa, J. Dannenhoffer, P. Wilcoxen, and C. Isik. Constrained Fuzzy Logic Approximation for Indoor Comfort and Energy Optimization. In *Fuzzy Information Processing Society, NAFIPS*, pages 500–504, 2005.

[23] M. Arora, S. Manne, I. Paul, N. Jayasena, and D. Tullsen. Understanding Idle Behavior and Power Gating Mechanisms in the Context of Modern Benchmarks on CPU-GPU Integrated Systems. In *Proc. of the Int'l Symp. on High Performance Computer Architecture (HPCA)*, 2015.

[24] M. Arora, S. Nath, S. Mazumdar, S. Baden, and D. Tullsen. Redefining the

Role of the CPU in the Era of CPU-GPU Integration. *IEEE Micro*, 32(6):4–16, Nov 2012.

[25] M. Awasthi, T. Suri, Z. Guz, A. Shayesteh, M. Ghosh, and V. Balakrishnan. System-Level Characterization of Datacenter Applications. In *Proc. of the Int'l Conf. on Performance Engineering (ICPE)*, pages 27–38, New York, NY, USA, 2015. ACM.

[26] P. Bailey, D. Lowenthal, V. Ravi, B. Rountree, M. Schulz, and B. de Supinski. Adaptive Configuration Selection for Power-Constrained Heterogeneous Systems. In *Proc. of the Int'l Conf. on Parallel Processing (ICPP)*, 2014.

[27] F. Bonomo, G. Durán, and J. Marenco. Exploring the Complexity Boundary between Coloring and List-coloring. *Annals of Operations Research*, 169(1):3–16, 2009.

[28] S. Boyd-Wickizer, R. Morris, and M. F. Kaashoek. Reinventing Scheduling for Multicore Systems. In *Proc. of the Conf. on Hot Topics in Operating Systems (HotOS)*, pages 21–21, Berkeley, CA, USA, 2009.

[29] L. Breiman. Random Forests. *Machine Learning*, 45(1):5–32, 2001.

[30] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. Classification and Regression Trees, 1999.

[31] R. H. Byers. *Half-Normal Distribution*. John Wiley & Sons, Ltd, 2005.

[32] B. Chai, A. Costa, S. Ahipasaoglu, S. Huang, C. Yuen, and Z. Yang. Minimizing Commercial Building Cost in Smart Grid: An Optimal Meeting Scheduling Approach. In *Proc. of the Int'l Conf. on Smart Grid Communications (SmartGridComm)*, pages 764–769, Nov. 2014.

[33] M. H. Chauhdry and P. B. Luh. Nested Partitions for Global Optimization in Nonlinear Model Predictive Control. *Control Engineering Practice*, 20(9):869 – 881, 2012.

[34] S. Che, B. M. Beckmann, S. K. Reinhardt, and K. Skadron. Pannotia: Understanding Irregular GPGPU Graph Applications. In *Proc. of the IEEE Int'l Symp. on Workload Characterization (IISWC)*, 2013.

[35] S. Che, M. Boyer, J. Meng, D. Tarjan, J. Sheaffer, S.-H. Lee, and K. Skadron.

Rodinia: A Benchmark Suite for Heterogeneous Computing. In *Proc. of the IEEE Int'l Symp. on Workload Characterization (IISWC)*, 2009.

[36] T. Chen, A. Rucker, and G. E. Suh. Execution Time Prediction for Energy-efficient Hardware Accelerators. In *Proc. of the Int'l Symp. on Microarchitecture (MICRO)*, pages 457–469. ACM, 2015.

[37] R. Cochran, C. Hankendi, A. K. Coskun, and S. Reda. Pack & Cap: Adaptive DVFS and Thread Packing Under Power Caps. In *Proc. of the Int'l Symp. on Microarchitecture (MICRO)*, 2011.

[38] C. Corbin and G. Henze. Predictive Control of Residential HVAC and its Impact on the Grid. Part I: Simulation Framework and Models. *Journal of Building Performance Simulation*, 10(3):294–312, 2017.

[39] C. Cortes and V. Vapnik. Support-vector Networks. *Machine Learning*, 20(3):273–297, Sep 1995.

[40] M. Curtis-Maury, J. Dzierwa, C. D. Antonopoulos, and D. S. Nikolopoulos. Online Power-performance Adaptation of Multithreaded Programs Using Hardware Event-based Prediction. In *Proc. of the Int'l Conf. on Supercomputing (ICS)*, 2006.

[41] M. Curtis-Maury, A. Shah, F. Blagojevic, D. S. Nikolopoulos, B. R. de Supinski, and M. Schulz. Prediction Models for Multi-dimensional Power-Performance Optimization on Many Cores. In *Proc. of the Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, 2008.

[42] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford, V. Tipparaju, and J. S. Vetter. The Scalable Heterogeneous Computing (SHOC) Benchmark Suite. In *Proc. of the Workshop on General-Purpose Computation on Graphics Processing Units (GPGPU)*, 2010.

[43] J. Demšar, T. Curk, A. Erjavec, Črt Gorup, T. Hočevar, M. Milutinovič, M. Možina, M. Polajnar, M. Toplak, A. Starič, M. Štajdohar, L. Umek, L. Žagar, J. Žbontar, M. Žitnik, and B. Zupan. Orange: Data Mining Toolbox in Python. *Journal of Machine Learning Research*, 14:2349–2353, 2013.

[44] G. Dhiman and T. S. Rosing. Dynamic Power Management Using Machine Learning. In *Proc. of the Int'l Conf. on Computer-Aided Design (ICCAD)*, 2006.

[45] S. Di, D. Kondo, and W. Cirne. Characterization and Comparison of Cloud versus Grid Workloads. In *Proc. of the Int'l Conf. on Cluster Computing (CLUSTER)*, pages 230–238, Sept 2012.

[46] J. R. Dobbs and B. M. Hencey. Predictive HVAC Control Using a Markov Occupancy Model. In *Proc. of the American Control Conf. (ACC)*, 2014.

[47] R. H. Dodiera, G. P. Henzeb, D. K. Tillerb, and X. Guob. Building Occupancy Detection through Sensor Belief Networks. *Energy and Buildings*, 38(9):1033–1043, 2006.

[48] B. Dong and K. Lam. A Real-time Model Predictive Control for Building Heating and Cooling Systems based on the Occupancy Behavior Pattern Detection and Local Weather Forecasting. *Building Simulation*, 7(1):89–106, 2014.

[49] P. Dongara, L. Bircher, and J. Darilek. AMD Richland Client APU. Presented at Hot Chips, August 2013.

[50] B. Dorronsoro, S. Nesmachnow, J. Taheri, A. Y. Zomaya, E.-G. Talbi, and P. Bouvry. A Hierarchical Approach for Energy-efficient Scheduling of Large Workloads in Multicore Distributed Systems. *Sustainable Computing: Informatics and Systems*, 4(4):252 – 261, 2014.

[51] J. Du, W. Guo, and R. Wang. Tourism Room Occupancy Rate Prediction Based on Neural Network. In *Advances in Neural Networks ISNN 2007*, volume 4493 of *Lecture Notes in Computer Science*, pages 80–84. Springer Berlin Heidelberg, 2007.

[52] E. Ebrahimi, R. Miftakhutdinov, C. Fallin, C. J. Lee, J. A. Joao, O. Mutlu, and Y. N. Patt. Parallel Application Memory Scheduling. In *Proc. of the Int'l Symp. on Microarchitecture (MICRO)*, pages 362–373, New York, NY, USA, 2011.

[53] V. L. Erickson, M. Carreira-Perpiñán, and A. E. Cerpa. OBSERVE: Occupancy-Based System for Efficient Reduction of HVAC Energy. In *Proc. of the Int'l Conf. on Information Processing in Sensor Networks (IPSN)*, pages 258–269, 2011.

[54] V. L. Erickson and A. E. Cerpa. Occupancy Based Demand Response HVAC Control Strategy. In *Proc. of the ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings (BuildSys)*, pages 7–12. ACM, 2010.

[55] V. L. Erickson, Y. Lin, A. Kamthe, R. Brahme, A. Surana, A. E. Cerpa, M. D. Sohn, and S. Narayanan. Energy Efficient Building Environment Control Strategies Using Real-time Occupancy Measurements. In *Proc. of the ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings (BuildSys)*, pages 19–24, New York, NY, USA, 2009. ACM.

[56] J. Fang, M. Hadiuzzaman, E. Yin, and T. Z. Qiu. DynaTAM: An Online Algorithm for Performing Simultaneously Optimized Proactive Traffic Control for Freeways. *Canadian Journal of Civil Engineering*, 41(4):315–322, 2014.

[57] W. Feng, H. Lin, T. Scogland, and J. Zhang. OpenCL and the 13 Dwarfs: A Work in Progress. In *Proc. of the Int'l Conf. on Performance Engineering (ICPE)*, 2012.

[58] P. C. Fishburn. *Interval Orders and Interval Graphs: A Study of Partially Ordered Sets*. Wiley New York, 1985.

[59] R. Fortet. Applications de l'algèbre de Boole en recherche opérationnelle. *Revue Française d'Informatique et de Recherche Opérationnelle*, 4:17–26, 1960.

[60] R. Z. Freire, G. H. Oliveira, and N. Mendes. Predictive Controllers for Thermal Comfort Optimization and Energy Savings. *Energy and Buildings*, 40(7):1353 – 1365, 2008.

[61] N. Fumo, P. Mago, and R. Luck. Methodology to Estimate Building Energy Consumption Using EnergyPlus Benchmark Models. *Energy and Buildings*, 42(12):2331 – 2337, 2010.

[62] P. X. Gao and S. Keshav. SPOT: A Smart Personalized Office Thermal Control System. In *ACM e-Energy*, 2013.

[63] M. Ghasemazar. Robust Optimization of a Chip Multiprocessor's Performance Under Power and Thermal Constraints. In *Proc. of the Int'l Conf. on Computer Design (ICCD)*, 2012.

[64] S. Goyal, H. A. Ingley, and P. Barooah. Occupancy-based Zone-Climate Control for Energy-efficient Buildings: Complexity vs. Performance. *Applied Energy*, 106(C):209 – 221, 2013.

[65] J. L. Greathouse and M. Daga. Efficient Sparse Matrix-Vector Multiplica-

tion on GPUs Using the CSR Storage Format. In *Proc. of the Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, 2014.

[66] D. Greenberg, K. Pratt, B. Hencey, N. Jones, L. Schumann, J. Dobbs, Z. Dong, D. Bosworth, and B. Walter. Sustain: An Experimental Test Bed for Building Energy Simulation. *Energy and Buildings*, 58(3):44 – 57, 2013.

[67] J. Guerreiro, A. Ilic, N. Roma, and P. Tomas. Multi-Kernel Auto-Tuning on GPUs: Performance and Energy-Aware Optimization. In *Proc. of the Int'l. Conf. on Parallel, Distributed and Network-Based Processing (PDP)*, 2015.

[68] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 1994.

[69] J. L. Henning. SPEC CPU2006 Benchmark Descriptions. *SIGARCH Comput. Archit. News*, 34(4):1–17, Sept. 2006.

[70] S. Hong and H. Kim. An Integrated GPU Power and Performance Model. In *Proc. of the Int'l Symp. on Computer Architecture (ISCA)*, 2010.

[71] J. Hsu. Obama Orders Speedy Delivery of First Exascale Supercomputer. `http://spectrum.ieee.org/tech-talk/computing/hardware/obama-orders-speedy-delivery-of-first-exascale-supercomputer-`. Accessed: 08/16/2015.

[72] W.-L. Hsu and T.-H. Ma. Fast and Simple algorithms for Recognizing Chordal Comparability Graphs and Interval Graphs. *SIAM Journal on Computing*, 28(3):1004–1020, 1998.

[73] J. Hutchins, A. Ihler, and P. Smyth. Modeling Count Data from Multiple Sensors: A Building Occupancy Model. In *Proc. of the Int'l Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMPSAP)*, pages 241–244, Dec 2007.

[74] C. R. Inc. CVX: Matlab Software for Disciplined Convex Programming, version 2.0 beta. `http://cvxr.com/cvx`, Sept. 2012.

[75] C. Isci and M. Martonosi. Runtime Power Monitoring in High-End Processors: Methodology and Empirical Data. In *Proc. of the Int'l Symp. on Microarchitecture (MICRO)*, 2003.

[76] X. Jiang, B. Dong, and L. Sweeney. From Sensor Network To Social

Network- A Study On The Energy Impact In Buildings. In *NIPS Workshop Analyzing Networks and Learning with Graphs*, pages 1–7, Dec. 2009.

[77] O. Kayiran, A. Jog, M. Kandemir, and C. Das. Neither More Nor Less: Optimizing Thread-level Parallelism for GPGPUs. In *Proc. of the Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, 2013.

[78] W. Kleiminger. *Simulating the Energy Savings Potential in Domestic Heating Scenarios in Switzerland*. ETH Zürich, 2014.

[79] L. Klein, G. Kavulya, F. Jazizadeh, J.-y. Kwak, B. Becerik-Gerber, P. Varakantham, and M. Tambe. Towards Optimization of Building Energy and Occupant Comfort Using Multi-Agent Simulation. In *Proc. of the Int'l Symp. on Automation and Robotics in Construction (ISARC)*, 2011.

[80] T. Komoda, S. Hayashi, T. Nakada, S. Miwa, and H. Nakamura. Power Capping of CPU-GPU Heterogeneous Systems through Coordinating DVFS and Task Mapping. In *Proc. of the Int'l Conf. on Computer Design (ICCD)*, pages 349–356, Oct 2013.

[81] J. Krumm and A. J. B. Brush. Learning Time-Based Presence Probabilities. In *Proc. of the Int'l Conf. on Pervasive Computing (Pervasive)*, pages 79–96, Berlin, Heidelberg, 2011.

[82] A. Kusiak, F. Tang, and G. Xu. Multi-Objective Optimization of HVAC System with an Evolutionary Computation Algorithm. *Energy*, 36(5):2440 – 2449, 2011.

[83] A. Kusiak, G. Xu, and F. Tang. Optimization of an HVAC System with a Strength Multi-Objective Particle-Swarm Algorithm. *Energy*, 36(10):5935 – 5943, 2011.

[84] J.-y. Kwak, P. Varakantham, R. Maheswaran, Y.-H. Chang, M. Tambe, B. Becerik-Gerber, and W. Wood. TESLA: An Energy-saving Agent that Leverages Schedule Flexibility. In *Proc. of the Int'l Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, May 2013.

[85] K. P. Lam, M. Höynck, B. Dong, B. Andrews, Y. Chiou, R. Zhang, D. Benitez, J. Choi, et al. Occupancy Detection through an Extensive Environmental Sensor Network in an Open-Plan Office Building. *IBPSA Building Simulation*, pages 1452–1459, 2009.

[86] P. Lama, Y. Li, A. M. Aji, P. Balaji, J. Dinan, S. Xiao, Y. Zhang, W. Feng, R. Thakur, and X. Zhou. pVOCL: Power-Aware Dynamic Placement and Migration in Virtualized GPU Environments. In *Proc. of the Int'l. Conf. on Distributed Computing Systems (ICDCS)*, 2013.

[87] A. K. Laur. Modeling the Spatio-temporal Variability of Solar Radiation on Buildings: A Case Study of Lewis Hall. Master's thesis, University of Southern California, 2014.

[88] T. Le, H. L. Vu, Y. Nazarathy, B. Vo, and S. Hoogendoorn. Linear-Quadratic Model Predictive Control for Urban Traffic Networks (ISTTT). In *Proc. of the Int'l Symp. on Transportation and Traffic Theory*, 2013.

[89] B. C. Lee and D. M. Brooks. Accurate and Efficient Regression Modeling for Microarchitectural Performance and Power Prediction. In *Proc. of the Int'l Symp. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2006.

[90] J. Lee, P. P. Ajgaonkar, and N. S. Kim. Analyzing Throughput of GPGPUs Exploiting Within-Die Core-to-Core Frequency Variation. In *Proc. of the Int'l Symp. on Performance Analysis of Systems and Software (ISPASS)*, 2011.

[91] J. Lee and H. Kim. TAP: A TLP-aware Cache Management Policy for a CPU-GPU Heterogeneous Architecture. In *Proc. of the Int'l Symp. on High Performance Computer Architecture (HPCA)*, 2012.

[92] M. Lee, S. Song, J. Moon, J. Kim, W. Seo, Y. Cho, and S. Ryu. Improving GPGPU Resource Utilization Through Alternative Thread Block Scheduling. In *Proc. of the Int'l Symp. on High Performance Computer Architecture (HPCA)*, 2014.

[93] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi. GPUWattch: Enabling Energy Optimizations in GPGPUs. In *Proc. of the Int'l Symp. on Computer Architecture (ISCA)*, 2013.

[94] W. Li, G. Jin, X. Cui, and S. See. An Evaluation of Unified Memory Technology on NVIDIA GPUs. In *Proc. of the Int'l Symp. on Cluster, Cloud and Grid Computing (CCGrid)*, pages 1092–1098, May 2015.

[95] C. Liao, Y. Lin, and P. Barooah. Agent-based and graphical modeling of building occupancy. *Journal of Building Performance Simulation*, 5:5–25, Jan. 2012.

[96] B. Lim, H. Hijazi, S. Thiébaux, and M. van den Briel. *Online HVAC-Aware Occupancy Scheduling with Adaptive Temperature Control*, pages 683–700. Springer International Publishing, Cham, 2016.

[97] B. Lim, M. van den Briel, S. Thiébaux, S. Backhaus, and R. Bent. HVAC-Aware Occupancy Scheduling. In *AAAI Conf. on Artificial Intelligence (AAAI)*, Jan. 2015.

[98] B. Lim, M. van den Briel, S. Thiébaux, R. Bent, and S. Backhaus. Large Neighborhood Search for Energy Aware Meeting Scheduling in Smart Buildings. In *Integration of AI and OR Techniques in Constraint Programming*, volume 9075, pages 240–254. 2015.

[99] J. Löfberg. *Minimax Approaches to Robust Model Predictive Control*, volume 812. Linköping University Electronic Press, 2003.

[100] J. Lu, T. Sookoor, V. Srinivasan, G. Gao, B. Holben, J. Stankovic, E. Field, and K. Whitehouse. The Smart Thermostat: Using Occupancy Sensors to Save Energy in Homes. In *Proc. of the Conf. on Embedded Networked Sensor Systems (SenSys)*.

[101] P. B. Luh, J. Xu, E. Ni, and R. Karanam. An Optimization-based Approach for Facility Energy Management with Uncertainties. *IFAC Proceedings Volumes*, 35(1):443 – 448, 2002. 15th IFAC World Congress.

[102] C.-K. Luk, S. Hong, and H. Kim. Qilin: Exploiting Parallelism on Heterogeneous Multiprocessors with Adaptive Mapping. In *Proc. of the Int'l Symp. on Microarchitecture (MICRO)*, 2009.

[103] D. Lutz. *Guide To Meeting Management*, chapter 7, page 63. Convene, Sep. 2000.

[104] K. Ma, X. Li, W. Chen, C. Zhang, and X. Wang. GreenGPU: A Holistic Approach to Energy Efficiency in GPU-CPU Heterogeneous Architectures. In *Proc. of the Int'l Conf. on Parallel Processing (ICPP)*, pages 48–57, Sept 2012.

[105] M. Maasoumy, B. Moridian, M. Razmara, M. Shahbakhti, and A. Sangiovanni-Vincentelli. Online Simultaneous State Estimation and Parameter Adaptation for Building Predictive Control. In *Proc. of the Dynamic Systems and Control Conf. (DSCC)*, 2013.

[106] M. Maasoumy, M. Razmara, M. Shahbakhti, and A. S. Vincentelli. Han-

dling Model Uncertainty in Model Predictive Control for Energy Efficient Buildings. *Energy and Buildings*, 77:377 – 392, 2014.

[107] A. Majumdar, G. Wu, K. Dev, J. L. Greathouse, I. Paul, W. Huang, A.-K. Venugopal, L. Piga, C. Freitag, and S. Puthoor. A Taxonomy of GPGPU Performance Scaling. In *Proc. of the IEEE Int'l Symp. on Workload Characterization (IISWC)*, 2015.

[108] S. Mamidi, Y.-H. Chang, and R. Maheswaran. Improving Building Energy Efficiency with a Network of Sensing, Learning and Prediction Agents. In *Proc. of the Int'l Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 45–52, Richland, SC, 2012.

[109] A. Marquez, C. Gomez, P. Deossa, and J. Espinosa. Infinite Horizon MPC and Model Reduction Applied to Large Scale Chemical Plant. In *Proc. of the Robotics Symposium, Latin American and Colombian Conference on Automatic Control and Industry Applications (LARC)*, 2011.

[110] A. McLaughlin, I. Paul, J. L. Greathouse, S. Manne, and S. Yalamanchili. A Power Characterization and Management of GPU Graph Traversal. In *Workshop on Architectures and Systems for Big Data (ASBD)*, 2014.

[111] A. McLaughlin, J. Riedy, and D. A. Bader. Optimizing Energy Consumption and Parallel Performance for Static and Dynamic Betweenness Centrality using GPUs. In *Proc. of the High Performance Extreme Computing Conf. (HPEC)*, 2014.

[112] A. Merkel and F. Bellosa. Memory-aware Scheduling for Energy Efficiency on Multicore Processors. In *Proc. of the 2008 Conf. on Power Aware Computing and Systems (HotPower)*, pages 1–1, Berkeley, CA, USA, 2008.

[113] A. Merkel, J. Stoess, and F. Bellosa. Resource-conscious Scheduling for Energy Efficiency on Multicore Processors. In *Proc. of the European Conf. on Computer Systems (EuroSys)*, pages 153–166, New York, NY, USA, 2010.

[114] N. Meskin, H. Nounou, M. Nounou, A. Datta, and E. Dougherty. Output-feedback Model Predictive Control of Biological Phenomena Modeled by S-systems. In *Proc. of the American Control Conf. (ACC)*, pages 1979–1984, June 2012.

[115] S. Meyn, A. Surana, Y. Lin, S. M. Oggianu, S. Narayanan, and T. A. Frewen. A Sensor-Utility-Network Method for Estimation of Occupancy

in Buildings. In *Proc. of the Int'l Conf. on Decision and Control (CDC)*, pages 1494–1500. IEEE, 2009.

[116] J. Murphy. Using Time-of-Day Scheduling To Save Energy. *Americal Society for Heating, Refrigerating, and Air-Conditioning Engineers (ASHRAE) Journal*, 2009.

[117] N. Nassif, S. Kajl, and R. Sabourin. Evolutionary Algorithms for Multi-Objective Optimization in HVAC System Control Strategy. In *Fuzzy Information, 2004*, volume 1, pages 51–56, 2004.

[118] R. Nathuji, C. Isci, and E. Gorbatov. Exploiting Platform Heterogeneity for Power Efficient Data Centers. In *Proc. of the Int'l Conf. on Autonomic Computing (ICAC)*, pages 5–5, June 2007.

[119] M. Nowak and A. Urbaniak. Utilization of Intelligent Control Algorithms for Thermal Comfort Optimization and Energy Saving. In *Proc. of the Int'l Carpathian Control Conf. (ICCC)*, pages 270–274, 2011.

[120] S. Nussbaum. AMD "Trinity" APU. Presented at Hot Chips, August 2012.

[121] F. Oldewurtel, A. Parisio, C. N. Jones, D. Gyalistras, M. Gwerder, V. Stauch, B. Lehmann, and M. Morari. Use of Model Predictive Control and Weather Forecasts for Energy Efficient Building Climate Control. *Energy and Buildings*, 45(0):15 – 27, 2012.

[122] J. Page, D. Robinson, N. Morel, and J.-L. Scartezzini. A Generalised Stochastic Model for the Simulation of Occupant Presence. *Energy and Buildings*, 40(2):83 – 98, 2008.

[123] S. Páll, M. J. Abraham, C. Kutzner, B. Hess, and E. Lindahl. Tackling Exascale Software Challenges in Molecular Dynamics Simulations with GROMACS. *CoRR*, 1506.00716, 2015.

[124] R. Palomar, J. Gómez-Luna, F. A. Cheikh, J. Olivares-Bueno, and O. J. Elle. High-Performance Computation of Bézier Surfaces on Parallel and Heterogeneous Platforms. *Int'l Journal of Parallel Programming*, May 2017.

[125] D. Pan, D. Wang, J. Cao, Y. Peng, and X. Peng. Minimizing Building Electricity Costs in a Dynamic Power Market: Algorithms and Impact on Energy Conservation. In *Real-Time Systems Symp. (RTSS)*, pages 107–117, Dec. 2013.

[126] D. Pan, Y. Yuan, D. Wang, X. Xu, Y. Peng, X. Peng, and P.-J. Wan. Thermal Inertia: Towards An Energy Conservation Room Management System. In *INFOCOM*, pages 2606 –2610, Mar. 2012.

[127] A. Pathania, Q. Jiao, A. Prakash, and T. Mitra. Integrated CPU-GPU Power Management for 3D Mobile Games. In *Proc. of the Design Automation Conf. (DAC)*, pages 40:1–40:6, New York, NY, USA, 2014.

[128] I. Paul, W. Huang, M. Arora, and S. Yalamanchili. Harmonia: Balancing Compute and Memory Power in High Performance GPU. In *Proc. of the Int'l Symp. on Computer Architecture (ISCA)*, 2015.

[129] I. Paul, S. Manne, M. Arora, W. L. Bircher, and S. Yalamanchili. Cooperative Boosting: Needy Versus Greedy Power Management. In *Proc. of the Int'l Symp. on Computer Architecture (ISCA)*, 2013.

[130] I. Paul, V. Ravi, S. Manne, M. Arora, and S. Yalamanchili. Coordinated Energy Management in Heterogeneous Processors. In *Proc. of the Conf. on High Performance Computing, Networking, Storage and Analysis (SC)*, 2013.

[131] A. Phansalkar, A. Joshi, and L. K. John. Subsetting the SPEC CPU2006 Benchmark Suite. *SIGARCH Comput. Archit. News*, 35(1):69–76, Mar. 2007.

[132] V. Putta, G. Zhu, D. Kim, J. Hu, and J. E. Braun. A Distributed Approach to Efficient Model Predictive Control of Building HVAC Systems. In *Proc. of the Int'l High Performance Buildings Conf. (IHPBC)*, Purdue University, IN, USA, 2012.

[133] J. R. Quinlan. Induction of Decision Trees. *Machine Learning*, 1(1):81–106, Mar. 1986.

[134] M. M. Rafique, N. Ravi, S. Cadambi, A. R. Butt, and S. Chakradhar. Power Management for Heterogeneous Clusters: An Experimental Study. In *Proc. of the Int'l Green Computing Conf. (IGCC)*, pages 1–8, July 2011.

[135] V. T. Ravi, M. Becchi, W. Jiang, G. Agrawal, and S. Chakradhar. Scheduling Concurrent Applications on a Cluster of CPU-GPU Nodes. In *Proc. of the Int'l Symp. on Cluster, Cloud and Grid Computing (CCGRID 2012)*, pages 140–147, May 2012.

[136] S. Rennich. CUDA C/C++ Streams and Concurrency.

[137] E. Rotem. Intel Architecture, Code Name Skylake Deep Dive: A New Architecture to Manage Power Performance and Energy Efficiency. Presented at Intel Developer Forum, August 2015.

[138] E. Rotem, R. Ginosar, C. Weiser, and A. Mendelson. Energy Aware Race to Halt: A Down to EARtH Approach for Platform Energy Management. *Computer Architecture Letters*, 13(1):25–28, Jan–June 2012.

[139] E. Rotem, A. Naveh, D. Rajwan, A. Ananthakrishnan, and E. Weissmann. Power-Management Architecture of the Intel Microarchitecture Code-Named Sandy Bridge. *IEEE Micro*, 32(2):20–27, March 2012.

[140] S. Royer, S. Thil, T. Talbert, and M. Polit. A Procedure for Modeling Buildings and their Thermal Zones Using Co-simulation and System Identification. *Energy and Buildings*, 78, Aug. 2014.

[141] T. R. W. Scogland, B. Rountree, W.-c. Feng, and B. R. de Supinski. Heterogeneous Task Scheduling for Accelerated OpenMP. In *Proc. of the Int'l Parallel and Distributed Processing Symposium (IPDPS)*, pages 144–155, Washington, DC, USA, 2012. IEEE Computer Society.

[142] J. Scott, A. Bernheim Brush, J. Krumm, B. Meyers, M. Hazas, S. Hodges, and N. Villar. PreHeat: Controlling Home Heating Using Occupancy Prediction. In *Proc. of the Int'l Conf. on Ubiquitous Computing (UbiComp)*, pages 281–290, New York, NY, USA, 2011.

[143] A. Sethia and S. Mahlke. Equalizer: Dynamic Tuning of GPU Resources for Efficient Execution. In *Proc. of the Int'l Symp. on Microarchitecture (MICRO)*, 2014.

[144] T. Sharmin, M. Gül, and M. Al-Hussein. A Data Analysis Framework for Optimizing Occupant Energy Use While Sustaining Indoor Environmental Quality. In *Proc. of the Int'l Construction Specialty Conf. of the Canadian Society for Civil Engineering (ICSC)*, Jun 2015.

[145] T. Sharmin, M. Gül, and M. Al-Hussein. A User-centric Space Heating Energy Management Framework for Multi-family Residential Facilities Based on Occupant Pattern Prediction Modeling. *Building Simulation*, May 2017.

[146] S. Shen, V. V. Beek, and A. Iosup. Statistical Characterization of Business-Critical Workloads Hosted in Cloud Datacenters. In *Proc. of the Int'l Symp. on Cluster, Cloud and Grid Computing (CCGrid)*, pages 465–474, May 2015.

[147] G. Singla, G. Kaur, A. K. Unver, and U. Y. Ogras. Predictive Dynamic Thermal and Power Management for Heterogeneous Mobile Platforms. In *Proc. of the Design, Automation and Test in Europe & Exhibition (DATE)*, pages 960–965, San Jose, CA, USA, 2015.

[148] S. Soner and C. Özturan. Integer Programming Based Heterogeneous CPU-GPU Cluster Scheduler for SLURM Resource Manager. In *Proc. of the Int'l Conf. on High Performance Computing and Communication (HPCC)*, pages 418–424, June 2012.

[149] J. A. Stratton, C. Rodrigues, I.-J. Sung, N. Obeid, L.-W. Chang, N. Anssari, G. D. Liu, and W. W. Hwu. Parboil: A Revised Benchmark Suite for Scientific and Commercial Throughput Computing. Technical Report IMPACT-12-01, University of Illinois at Urbana-Champaign, March 2012.

[150] B. Su, J. L. Greathouse, J. Gu, M. Boyer, L. Shen, and Z. Wang. Implementing a Leading Loads Performance Predictor on Commodity Processors. In *Proc. of the USENIX Annual Technical Conf. (USENIX ATC)*, 2014.

[151] B. Su, J. Gu, L. Shen, W. Huang, J. L. Greathouse, and Z. Wang. PPEP: Online Performance, Power, and Energy Prediction Framework and DVFS Space Exploration. In *Proc. of the Int'l Symp. on Microarchitecture (MICRO)*, 2014.

[152] S. Su, Q. Huang, J. Li, X. Cheng, P. Xu, and K. Shuang. Enhanced Energy-Efficient Scheduling for Parallel Tasks Using Partial Optimal Slacking. *The Computer Journal*, 58(2):246–257, 2015.

[153] G. E. Suh, L. Rudolph, and S. Devadas. Effects of Memory Performance on Parallel Job Scheduling. In *Revised Papers from the Int'l Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, pages 116–132, London, UK, UK, 2001.

[154] Y. Sun, X. Gong, A. K. Ziabari, L. Yu, X. Li, S. Mukherjee, C. Mccardwell, A. Villegas, and D. Kaeli. Hetero-mark, A Benchmark Suite for CPU-GPU Collaborative Computing. In *2016 IEEE Int'l Symp. on Workload Characterization (IISWC)*, pages 1–10, Sept 2016.

[155] R. Teodorescu and J. Torrellas. Variation-Aware Application Scheduling and Power Management for Chip Multiprocessors. In *Proc. of the 35th Annual Int'l Symp. on Computer Architecture (ISCA)*, pages 363–374, Washington, DC, USA, 2008.

[156] E. Totoni, J. Torrellas, and L. V. Kale. Using an Adaptive HPC Runtime System to Reconfigure the Cache Hierarchy. In *Proc. of the Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, 2014.

[157] R. Van Der Linden and A. P. Leemhuis. The Use of Model Predictive Control for Asset Production Optimization: Application to a Thin-Rim Oil Field Case. In *SPE Annual Technical Conference and Exhibition*. Society of Petroleum Engineers, 2010.

[158] A. Vilches, R. Asenjo, A. Navarro, F. Corbera, R. Gran, and M. Garzarn. Adaptive Partitioning for Irregular Applications on Heterogeneous CPU-GPU Chips. *Procedia Computer Science*, 51:140 – 149, 2015.

[159] E. Vrettos, K. Lai, F. Oldewurtel, and G. Andersson. Predictive Control of Buildings for Demand Response with Dynamic Day-ahead and Real-time Prices. In *2013 European Control Conf. (ECC)*, pages 2527–2534, July 2013.

[160] L. Vu, Q. Do, and K. Nahrstedt. Jyotish: Constructive Approach for Context Predictions of People Movement from Joint Wifi/Bluetooth Trace. *Pervasive Mobile Computing*, 7(6):690–704, Dec. 2011.

[161] D. Wang, C. C. Federspiel, and F. Rubinstein. Modeling Occupancy in Single Person Offices. *Energy and Buildings*, 37(2):121–126, 2005.

[162] H. Wang, V. Sathish, R. Singh, M. J. Schulte, and N. S. Kim. Workload and Power Budget Partitioning for Single-chip Heterogeneous Processors. In *Proc. of the Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, 2012.

[163] P.-H. Wang, C.-L. Yang, Y.-M. Chen, and Y.-J. Cheng. Power Gating Strategies on GPUs. *ACM Trans. on Architecture and Code Optimization*, 8(3):13:1–13:25, Oct. 2011.

[164] Y. Wang and S. Boyd. Fast Model Predictive Control Using Online Optimization. *IEEE Trans. on Control Systems Technology*, 18(2):267–278, March 2010.

[165] Y. Wang, S. Roy, and N. Ranganathan. Run-time Power-gating in Caches of GPUs for Leakage Energy Savings. In *Proc. of the Int'l. Design, Automation Test in Europe Conference & Exhibition (DATE)*, 2012.

[166] Z. Wang, L. Zheng, Q. Chen, and M. Guo. CPU+GPU Scheduling with Asymptotic Profiling. *Parallel Computing*, 40(2):107 – 115, 2014.

[167] J. A. Winter, D. H. Albonesi, and C. A. Shoemaker. Scalable Thread Scheduling and Global Power Management for Heterogeneous Many-core Architectures. In *Proc. of the Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, pages 29–40, New York, NY, USA, 2010.

[168] C. Wren, Y. Ivanov, D. Leigh, and J. Westhues. The MERL Motion Detector Dataset: 2007 Workshop on Massive Datasets. Technical report, 2007.

[169] C.-M. Wu, R.-S. Chang, and H.-Y. Chan. A Green Energy-Efficient Scheduling Algorithm Using the DVFS Technique for Cloud Datacenters. *Future Generation Computer Systems*, 37:141 – 147, 2014.

[170] G. Wu, J. L. Greathouse, A. Lyashevsky, N. Jayasena, and D. Chiou. GPGPU Performance and Power Estimation Using Machine Learning. In *Proc. of the Int'l Symp. on High Performance Computer Architecture (HPCA)*, 2015.

[171] D. Yan, W. OBrien, T. Hong, X. Feng, H. B. Gunay, F. Tahmasebi, and A. Mahdavi. Occupant Behavior Modeling for Building Performance Simulation: Current State and Future Challenges. *Energy and Buildings*, 107:264 – 278, 2015.

[172] Z. Yang, N. Li, B. Becerik-Gerber, and M. Orosz. A Multi-sensor Based Occupancy Estimation Model for Supporting Demand Driven HVAC Operations. In *Proc. of the Symp. on Simulation for Architecture and Urban Design (SimAUD)*, pages 2:1–2:8, San Diego, CA, USA, 2012.

[173] Z. Yang, N. Li, B. Becerik-Gerber, and M. Orosz. A Systematic Approach to Occupancy Modeling in Ambient Sensor-rich Buildings. *SIMULATION*, 2013.

[174] W. Yuan and K. Nahrstedt. Energy-efficient Soft Real-time CPU Scheduling for Mobile Multimedia Systems. In *Proc. of the ACM Symp. on Operating Systems Principles (SOSP)*, pages 149–163, New York, NY, USA, 2003.

[175] X. Zhan and S. Reda. Techniques for Energy-efficient Power Budgeting in Data Centers. In *Proc. of the Design Automation Conf. (DAC)*, pages 176:1–176:7, New York, NY, USA, 2013.

[176] C. Zhang, A. Ravindran, K. Datta, A. Mukherjee, and B. Joshi. A Machine Learning Approach to Modeling Power and Performance of Chip Multiprocessors. In *Proc. of the Int'l Conf. on Computer Design (ICCD)*, 2011.

[177] Z. Zhang, M. Lang, S. Pakin, and S. Fu. Trapped Capacity: Scheduling Under a Power Cap to Maximize Machine-room Throughput. In *Proc. of the Int'l Workshop on Energy Efficient Supercomputing (E2SC)*, 2014.

[178] Y. Zhu and V. J. Reddi. High-performance and Energy-Efficient Mobile Web Browsing on Big/Little Systems. In *Proc. of the Int'l Symp. on High Performance Computer Architecture (HPCA)*, 2013.

[179] Y. Zong, L. Mihet-Popa, D. Kullmann, A. Thavlov, O. Gehrke, and H. Bindner. Model Predictive Controller for Active Demand Side Management with PV Self-consumption in an Intelligent Building. In *Proc. of the Int'l Conf. and Exhibition on Innovative Smart Grid Technologies (ISGT)*, pages 1–8, Oct 2012.